第1章 基　　础

1.1 术　　语

发送者和接收者

在信息传送过程中，主动提供信息的一方称发送者，得到信息的一方称接收者。发送者要确保信息安全传送，使窃听者不能盗取信息。

信息和加密

原始信息是清楚明白的文本，它又称明文。以隐藏信息实质内容的方式伪装信息的过程就是加密，加密后的信息称为密文。把密文转回明文的过程称解密。所有这一切均表示在图1.1中。

保持信息安全的科学技术称为密码术(Cryptography)，它是密码员的工作内容。揭示密文伪装的科学技术称为密码分析或密码破译(为行文方便，本书两种称呼叠用)，它是密码破译员(或密码分析员)的工作内容。兼有密码术和密码破译两方面内容的数学分支就是密码学(Cryptology)，这是密码学家的研究领域。密码学学科要求近代密码学家应具有理论数学方面的训练。

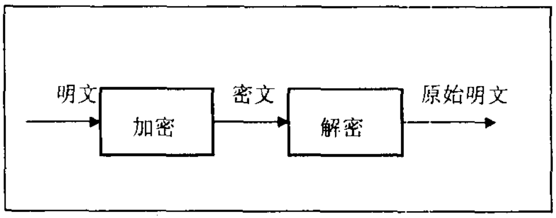


图1.1　加密和解密

明文信息用M(Message)或P(Plaintext)表示。它是比特流、一个文本文件、一个位信息图、数字化语音流、或数字化视频图像等类似物。只要联系到计算机，M就是简单的二进制数据。

密文用C(Ciphertext)表示。它也是二进制数据。有时与M等长，有时大于(压缩再加密，C可能会小于M，然而，单加密通常不能做到这样)。加密函数E作用于M产生了C，数学表示为：

$$E(M)=C$$

逆操作，解密函数D作用于C产生了M：

$$D(C)=M$$

因为加密和解密信息的全过程是恢复原始明文，下列表达式必须成立：

$$D(E(M))=M$$

认证，完整性和不许抵赖

密码术除了提供机密性之外，还要求做以下工作：

认证：信息接受者应有可能核查信息来源，任何闯入者无力装扮成原有的发送者。

完整性：信息接收者应有可能验证信息在传播中未经修改，闯入者无力用假信息代替合法的原有信息。

不可抵赖：发送者在发送出信息后无法抵赖他所发出的信息。

这些计算机上进行社交的要点也类似于人们面对面交流时的要求。某人讲他是何许人时

就需用证件(不管是工作证,驾驶执照或护照)来证实。这些表示某人身份的证件确能代表其人。认证,完整性和不可抵赖提供的就是这类服务。

算法和密钥

密码算法,称为加密算法,是用来加密和解密的数学函数。为给明文信息加密,用加密算法函数。为给密文信息解密,用解密算法函数。

如果算法的安全性基于算法工作的保安,这是一个受约算法。受约算法只是历史上曾被注意,可惜不适合今日的要求。大的或变动着的用户团体不能使用它们,因为每次一个用户离开了,团体中的每一个人必须改换不同的算法。如果团体中某个人偶然泄漏了秘密,其他每一个人也必须更换算法。

更糟的是受约算法无质量控制和标准。每一个用户团体必须有他们自己独特的算法。他们不能用现成的硬件或软件产品,因为窃听者可以购买同一商品并弄清算法,从而不得不设计自己的算法并付之实现。如果该团体中没有很好的密码员,他们将不清楚所用的算法是否安全。尽管这样,对低安全性的应用来讲,受约算法还是非常有效的。Zenith 的视频编码技术是一个加密算法的例子。

近代密码术用密钥来解决安全性问题。密钥用 K 表示,它可以是任何一个大数。这个密钥能取许多值中的一个。密钥的取值范围称为密钥空间。

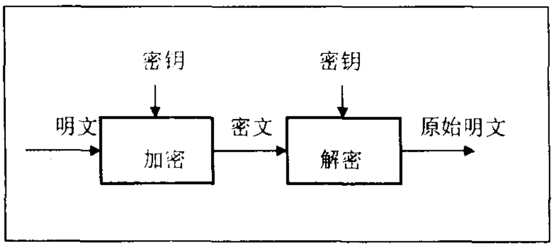


图 1.2 具有相同加密和解密密钥的系统

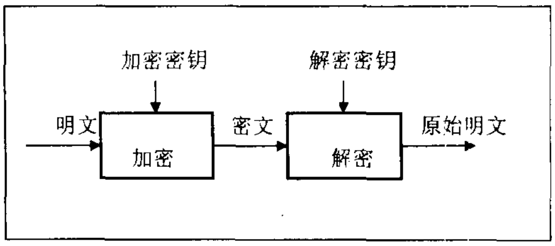


图 1.3 具有二个密钥的加密和解密系统

密钥可同时用于加密和解密,那么加密和解密函数表示为:

$$E_K(M)=C$$

$$D_K(C)=M$$

如果加密密钥和解密密钥是相同的,则这两个函数具有以下性质:

$$D_K(E_K(M))=M$$

如图 1.2 所示。

某些算法用不同的加密密钥和解密密钥(见图 1.3)。即,加密密钥 K_1,解密密钥 K_2。在这种情况下:

$$E_{K_1}(M)=C$$

$$D_{K_2}(C)=M$$

$$D_{K_2}(E_{K_1}(M))=M.$$

密钥决定了这些算法的安全性,而不是算法的内容。所以人们尽管可以发表和分析算法,用算法可以制造大量商品,即使窃听者知道算法,但若不知特有的密钥仍不能读取信息。一个密码系统应包括算法以及所有可能的明文、密文和密钥。

对称算法和公开密钥算法

基于密钥的算法有两种一般形式:对称的和公开密钥。对称算法,也称常用算法,是加密密

钥能从解密密钥求得或解密密钥能从加密密钥求得的算法。在许多对称系统中，加密密钥和解密密钥是相同的。这种算法，也称为保密密钥算法、单密钥算法或一密钥算法，需要发送者和接收者在他们通信之前协商密钥。这种密钥必须保密。对称算法的安全性依赖于密钥；泄密的密钥意味着任何人在这个加密系统中能加密和解密信息。

用对称算法的加密和解密表示为：

$$E_K(M)=C$$

$$D_K(C)=M$$

对称算法能划分为两类。一类称为流算法或流加密算法，另一类则是按组进行，在明文中每次操作一组比特。这些比特组称为块。这类算法称为块算法或块加密算法。对计算机上实施的算法，典型的块大小为 64 比特，大到可阻止破译，小到易于操作。在块和流算法中，加密和解密中使用相同的密钥(在用计算机之前，算法对明文通常每次操作一个字符。可认为这是字符流操作算法，也可认为 8 比特块的块操作算法。)。

公开密钥算法也称不对称算法。他们的设计使得应用于加密的密钥不同于应用解密的密钥。进而，解密密钥不能(至少不能在有限计数之内)从加密密钥中计算出。因为加密密钥为公开的，所以称为公开密钥系统：一个完全陌生的人能使用加密密钥加密信息，但仅拥有相应解密密钥者才能解密信息。在这个系统中，加密密钥常称为公开密钥，简称公钥。而解密密钥称为私有密钥或秘密密钥，简称私钥。

使用公钥 K 加密表示为：

$$E_K(M)=C$$

尽管公钥和私钥是不同的，但具有相应私钥的解密表示为：

$$D_K(C)=M$$

有时，信息将用私钥加密和用公钥解密，如数字签名。尽管可能混淆，这些操作仍分别表示为：

$$E_K(M)=C$$

$$D_K(C)=M$$

密码破译

密码术的主要目的是考虑到窃听者能完全拾取收发二者之间的通信信息情况下，实现安全信息传送。

密码破译则是无须取得密钥而能将密文信息恢复成明文的科学。密码破译成功，不仅恢复明文也可得到密钥。并且发现密码系统中导致上述结果的弱点。

一次破译也称一次攻击。破译中的基本假设是保安性全取决于密钥而破译员了解加密算法和实现的全部细节。实际上并非完全如此，这只是理想的假定。

有四种一般类型的攻击方法。当然他们中的每一种都假定破译员已有所用加密算法的全部知识。

1. 仅知密文攻击

在这种攻击中，密码破译员有几个信息的密文，所有的信息使用相同加密算法加密。为将相同密钥加密的信息解密，密码破译员会尽可能多地恢复信息的明文，或者更好地是导出用于加密信息的密钥。

已知： $C_1=E_k(P_1), C_2=E_k(P_2), \cdots \quad C_i=E_k(P_i)$

导出： $P_1、P_2 \cdots P_i、k$

或导出从 $C_{i+1}=E_k(P_{i+1})$ 得知 P_{i+1} 的算法

2. 已知明文攻击

密码破译员不但利用几个信息的密文，而且有这些信息的明文。导出用于加密信息的密钥或用相同的密钥加密任何新的信息的解密算法。

已知： $P_1, C_1=E_k(P_1), P_2, C_2=E_k(P_2), \quad \cdots \quad P_i, C_i=E_k(P_i)$

导出： k 或导出从 $C_{i+1}=E_k(P_{i+1})$ 得知 P_{i+1} 的算法

3. 选择明文攻击

密码破译员不但利用密文和涉及到的信息的明文，而且能选择加密的明文。这比已知明文攻击更强有力，因为密码破译员能选择专用的明文块加密，可能产生更多关于密钥的信息。导出用于加密信息的密钥或用相同的密钥加密任何新的信息的解密算法。

已知： $P_1, C_1=E_k(P_1), P_2, C_2=E_k(P_2), \quad \cdots \quad P_i, C_i=E_k(P_i)$

其中密码分析者选择 P_1、P_2 ··· P_i

导出： k 或导出从 $C_{i+1}=E_k(P_{i+1})$ 得知 P_{i+1} 的算法

4. 自适应选择明文攻击

这是选择明文攻击的一个特例。密码破译员不但选择加密的明文，而且能依据以前加密的结果修改选择。在选择明文攻击中，密码破译员可以选择一大块明文加密；在自适应选择明文攻击中，选择一个较小的明文块，然后依据第一次的结果选择另一块，等等。

5. 选择密文攻击

密码破译员能选择不同的将要解密的密文和已解的明文，将它们放入一个自动解密的分解判别的工具箱中，导出密钥。

已知： $C_1, P_1=D_k(C_1), C_2, P_2=D_k(C_2), \quad \cdots \quad C_i, P_i=D_k(C_i)$

导出： k

这种攻击主要应用于公开密钥加密系统。选择密文攻击有时对对称算法一样有效。

6. 选择密钥攻击

这种攻击并不意味破译员能选择密钥。只不过破译员对不同密钥之间的关系有所了解。它较新奇和含混但并不实用。

7. 牛皮筋式破译

破译员威胁、写黑信或折磨某人直至给出密钥，或用金钱收买，所以也称购买密钥攻击，这已不是科学技术的内容了，但在现实社会中却也常有奏效。

已知明文和选择明文攻击比想象中用得更普遍。许多信息有密码破译员熟知的标准开始和结尾。源代码加密后仍易受攻击，因为规则地出现一些关键词如 #define、struct、else、return。加密可执行代码也有同样问题，如函数、循环结构等等。已知明文攻击成功的例子，在二次大战时对日本和德国的作战中都有记录。

讨论攻击时不要忘记我们的基本假定，破译员是熟知算法的。把密码系统的强度建立在算法保密上的想法是错误的。最好的算法，被世界级破译员攻击多年而仍未攻破的算法都已公开，还有什么可值得保密的算法呢？

密码破译员并不总有获得算法的机会，如二次大战中美国破译日本外交代码 purple 就是这样，但他们经常会得到算法。如果算法是使用在商业的保密程序中，花钱买反汇编程序就可揭开算法。如果算法是用于军事通讯系统，也很简单，花钱买设备或窃听然后恢复算法。

那些因暂时未被破译就简单地宣布拥有不可破译密码的人，不是天才就是傻瓜，不幸世界

上多是后者。夸耀算法优点但又拒绝公开，相信自己的算法万能的人，要当心。好的密码员会依靠同行评论区分出好的和坏的算法。

算法安全性

算法根据攻破的难易标定了不同的安全程度。如攻破某算法的费用大于加密数据的代价，该算法认为是安全的。如攻破某算法的时间长于数据的保密期，该算法也认为是安全的。同样如攻破某算法所需的数据量多于密钥加密的数据量，该算法仍认为是安全的。以上所说的安全并非绝对，因为在密码破译中新的攻破机遇频频发生，另一方面大部分数据的价值会随时间流逝而贬低。所以，安全的要点是数据的价值要继续保持低于攻破安全屏障所花的代价。

Lars Knudsen 按攻破算法难易的递降顺序，分类如下：

· 全攻破，密码破译员找到密钥 K，于是解密 $D_K(C)=M$。

· 全局推导，密码破译员找到另一算法 A，无须知道 K，因为 A 等价于 $D_K(C)$而解密。

· 局部推导，密码破译员找到窃听来的密文的明文而解密。

· 信息推导，密码破译员得到关于密钥信息或是有关明文格式的信息。利用这些信息而解密。

如果无论破译员有多少密文，仍无足够信息能恢复明文，这样的算法是无条件安全的。事实上只有一次一用的密码本是不可攻破的。其它所有密码系统在惟密文攻击下都是可以攻破的。简单的做法就是尝试所有可能的密钥，然后判断明文的意义，这也称蛮力攻击。

密码术更关心密码系统不能被计算机攻破，一算法认为是“计算”性地安全，指在有效资源下，无论现在或将来都是不能攻破的。后文将解释什么是“有效资源”。

也可以不同方式度量攻击的复杂性：

· 数据复杂性，为攻击所需数据的攻击量。

· 处理复杂性，执行攻击所须的时间，又称工作因子。

· 存储需求，做攻击所须的存储量。

攻击的复杂性取这些因素中的最小值。某些攻击涉及到三个因素的平衡折衷。较快的攻击往往花费较大的存储需求。

复杂性往往用数量级表示，如果算法处理的复杂度为 2^{128}，则攻破算法需要 2^{128}次操作，假定计算速度为 100 万次/秒，并由 100 万个这样的处理机来并行处理，还须 10^{17}年才能揭示密钥。这是宇宙年龄的 10 亿倍。

然而攻击复杂性是恒定的，而计算能力却不同，在本世纪的后半世纪它有着显著的进步而且没有理由认为会停止。平行机很适合用来作密码破译攻击。攻击任务被分成亿万个小片，处理机间不需交互作用。总之，简单地将因为现有技术的局限而未能攻破的算法宣称为绝对安全算法是危险的。设计好的密码系统时，必须预期计算能力经若干年的发展后，仍然是不可破译。

1.2 密写术

密写术是把密件隐藏在其它信息中而不觉察存有秘密的书写方法。一般发送者用一大片无关信息写在纸上然而把密件隐藏其中。过去采用过的技巧有不可见墨水，在选用的字符上带小针孔，手写字符间的微小差异，打印出来的字符上用铅笔做标记，格孔密写卡(将它覆盖在一张纸上从格孔中写入密件，然后在纸上余下部分填入其它字句，使它像一般信件)等等。

近来人们常把密件藏于图形、图像中，将图像信息每一字节的最低位替换成密件信息位。这样加工后的图像并无肉眼可知的变化，而密件却能安全地到达接受方。一个 1024×1024 灰度级的图像用这样的方法能存储 64K 字节的信息。有一些公开的用来做这种密写的加工程序。

Peter Wayner's 模拟函数(mimic functions)可将密件信息搅乱,这些函数修饰密件,使其统计面貌相似报纸分类内容,莎士比亚的剧本,互联网络的新闻。这种密写方法只对计算机扫视信息时奏效,而不能蒙混人眼。

1.3 替代加密和转移加密

在计算机出现之前,密码学包含的是基于字符的各种算法。不同的密码算法或是一字符替代成其他字符,或是一字符转置成其他字符。较好的算法则二者都进行并多次反复。

目前,事情更为复杂,但道理是一样的。主要的变化是算法实施于每一个二进位而不是一个字符。实际上这不过是"字母表"的大小有所变化:26 个元素变成 2 个元素。多数好的加密算法仍是把元素结合成一定大小进行替代和转置。

替代加密算法

替代加密算法是明文中的每一个字符用另一个字符替代形成密文。除接受者外其他人不理解其间的替代。接受者对密文作反向替代后恢复成明文。

在经典密码学中,替代加密算法有四种基本类型:

1. 简单替代加密算法或称单字符加密,明文每一字符被替代成密文中的相应字符。新闻密电就是用这种方法。

2. 同音替代加密算法,类似于简单替代加密算法,但明文中的单一字符能对应密文中的几个字符。例如"A"可能相应于 5、13、25 或"B"可能相应于 7、19、31 或 42 等等。

3. 多元替代加密算法,成块的字符加密成一组其它字符。如"ABA"相应于"RTQ","ABB"相应于"SLL"等等。

4. 多字母替代加密算法,由多次简单替代加密组成。例如用 5 次不同的简单替代加密,具体所用的次数随每一字符在明文中的位置而变化。

著名的凯撒加密算法是将每一明文字符用右移三位并以 26 为模的字符所替代(A 用 D 替代,B 用 E……W 用 Z、X 用 A、Y 用 B、Z 用 C 替代)。它是一个简单的加密算法。

ROT13 也是一个简单加密程序,通常可在 Unix 系统中找到它,每一字符移动 13 个位置,如 A 用 N 替代,B 用 O 替代等等。

用 ROT13 两次加密一个文件,则恢复成原文件。

M=ROT13(ROT13(M))

ROT13 并不着眼于安全性,它常用在网上公布一些带有冒犯性的文章以费人猜疑。

简单替代加密易于攻破,因为加密算法并不涉及明文不同字母的隐藏次数。密码破译者重建明文对每一个字母大约要试的是其它 25 个字母。

早在 1401 年,曼图亚的 Duchy,就采用了同音替代加密算法。它们比简单替代加密算法复杂得多,因而更难破译。但因为仍然没隐藏明文所有的统计特性,用已知明文攻击方法,很容易被攻破;用仅知密文攻击,会难一些,但对计算机也是几秒钟的事情。

多元替代加密算法将字母按组进行加密,例如英国在一次世界大战中采用的 playfair 算法。它是 1854 年发明的,一次加密一对字母。Hiu 算法也是一种多元替代加密算法。有时用 Hoffman 编码进行加密,那是不安全的多元替代加密。

在 1568 年,Leon Battisfa 发明的多字母替代加密算法,曾在美国内战中被北方军使用。尽管这种加密很容易被攻破,许多商业计算机安全产品都使用这种形式的算法(如 Word Perfect)。Vigenere 和 Beaufort 算法也是多字母替代加密算法的例子。

多字母替代加密算法有多个单字母密钥，每个密钥用来加密明文中的一个字母，第一个密钥加密第一个字母，第二个密钥加密明文中的第二个字母，依此类推。所有密钥用完之后，再循环使用。如果有 20 个单字母密钥，那么每 20 个字母将用相同的密钥加密。这称为密码的周期。在经典密码学中，周期长的密钥比周期短的密钥更难破译，但采用计算机技术，周期非常长的替代加密也能很容易被攻破。

运行密钥加密算法是这类算法的另一个例子。它用一个文本来加密另一个文本。尽管这样使密码的周期长达一个文本，但破译仍是轻而易举。

转置加密算法

转置加密后的密文有相同的字符长度，只是字符的次序改变了。简单的圆柱转置加密算法是将明文横着写在一张宽度固定的图纸上，然后垂直地读出即成密文，解密只不过是将密文竖着写在一张同样宽度的图纸上，垂直读出明文。如图 1.4 所示。

明文：COMPUTER GRAPHICS MAY BE SLOW BUT AT LEAST IT' S EXPENSIVE

```
COMPUTERGR
APHICSMAYB
ESLOWBUTAT
LEASTITSEX
PENSIVE
```

密文：CAELP OPSEE MHLAN PIOSS UCWTI TSBIU EMUTE RATSG YAERB TX

图 1.4 圆柱互换加密算法

因为密文和明文的字母完全一样，对密文的频率分析表明每个字母出现的可能性几乎相同。根据这个很好的提示，密码破译员用各种技术试探出字母的真正排列，从而获得明文。将密文再转置一次，会大大提高安全性。但无论多复杂的转置加密，计算机几乎都能攻破译。

在第一次世界大战中，德国采用的 ADFGX 加密算法，是一种转置加密算法，它同时又采用了简单的替代加密。当时这已经是一个复杂的算法了，但还是被一名法国的密码破译员 Georges Painvin 攻破。尽管许多现代加密算法都采用转置加密，费事的是它需要大量存储空间。但有时要求信息只有一定长度。因而替代加密更为普遍。

旋转机器

在 20 世纪 20 年代，为完成自动加密，各种机械加密装置应运而生。它们根据旋转的概念，用机械旋转来完成一般的替代。

旋转机器由一个键盘和一组旋子组成来实现 Vigenere 加密算法。每一个旋子有 26 个位置，刚好是字母表的一个任意排列，完成一个简单的替代。例如，一个旋子用“F”替换“A”、“U”替换“B”、“L”替换“C”等等。同时，一个旋子的输出端栓在另一个旋子的输入端。

比如，一个旋子机器由四个旋子组成，第一个旋子将“A”替换成“F”，第二个将“F”替换成“Y”，第三个将“Y”替换成“E”，最后一个将“E”替换成“C”，那么 C 就是输出的密文。如果，某几个旋子转动一下，下一次的替代又将不同于前一次。

旋子与齿轮的转动使得机器安全运行，因为每一个旋子的不同转速转动，由 n 个旋子组成的机器周期为 26^n，有些旋转机器的每一个旋子上位置的数量不同，这样更让密码破译员伤透脑筋。

最著名的旋转机器是Enigma，曾被德国在二次大战中使用。它是由欧洲的Arthur Scherbius和Arvid Gerhard Danin发明，Arthur Scherbius在美国取得了专利，德国人大大改进了最初的设计为战时使用。

德国的Enigma由三个旋子组成(从原来的五个中选出)，一块抽板(plugboard)用来稍微排序一下明文，一个反射旋子每个旋子对明文的每个字母操作两遍。Enigma已是相当复杂，但它在第二次世界大战中被攻破，第一次是被波兰的密码破译员攻破，密文是关于德国对英国的进攻。随着战争的发展，德国人进一步改进了Enigma但仍不断被英国破译。

1.4 简单的XOR

XOR是异或运算。C语言中用“^”表示，数学符号为⊕。它是一种标准的位运算：

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

又定义：

$a \oplus a = 0$

$a \oplus b \oplus b = a$

简单的XOR运算仅仅是一个Vigenere多字母加密算法而已。在这里提到它是因为它在商业软件包中甚为流行，至少MS-DOS和Macintosh就采用了。不幸的是，如果一个软件安全程序宣称它有一个“私有的”加密算法，甚至比OES快得多，那它只可能是下面算法的变体。

```
/* Usage: crypto key input_file output_file */
void main(int argc, char *argv[])
{
    FILE *fi, *fo;
    char *cp;
    int c;
    if ((cp=argv[1])&*cp! ='\0') {
        if ((fi=fopen(argv[2],"rb"))! =NULL) {
            if ((fo=fopen(argv[3],"wb"))! =NULL) {
                while ((c=getc(fi))! =EOF) {
                    if (! *cp) cp=argv[1];
                    c^ = *(cp++);
                    putc(c,fo);
                }
                fclose(fo);
            }
            fclose(fi);
        }
    }
}
```

这是一个对称算法，明文和密钥异或得到密文。因为将相同值异或两次就可得到原文，加密和解密因此可用完全一样的程序。

$$M \oplus K = C \qquad C \oplus K = M$$

这里没有真正的安全可言。这种加密，即使不用计算机也可轻而易举攻破。用计算机则是瞬间的事。

假设明文是英文，而且密钥长度是任意少的字节数。以下是破译过程：

• 首先，用“计算一致”步骤得到密钥长度。将密文与其自身移动不同位数后进行异或，计算相同的字节数。如果移动的位数是密钥的倍数，那么应该有超过 6%的相同字节，否则低于 0.4%的字节将不同(这里我们假设随机密钥加密的是一般的 ASCII 文本，别的明文应另当别论)。这个位置叫做一致下标。最短的并具有上述特性的移动距离就是密钥的长度。

• 移动密文，同自身异或。这样就消掉了密钥，留下的是明文和明文移动密钥长度后按上述长度异或的结果。因为每字节英文有 1.3 比特的实际信息，所以有足够的冗余决定一个解密的方法。

尽管这样，吹捧这种玩具算法几乎与 DES 一样安全的软件卖主多得令人吃惊！就是这种算法，NSA 最后允许美国数字电话工业用作声音隐私，XOR 加密算法也许可以防止你的小妹妹读你的私人信件，但无法阻挡密码破译员在几分钟内攻破。

1.5 一次一用密码本

信不信由你，有一个完美的加密方案叫做“一次一用密码本”。它是 1917 年由 Major Joseph Manborgne 和 AT&T 的 Gilbert Vernam 发明的。实际上，一次一用密码本是门槛方案的一个特殊情况。传统地说，一次一用密码本只不过是将许多不重复的随机的字母，写在几张纸上，然后粘贴成一本密码本。最初的形式，是电报打字员一次性胶带。发送者用本子上的每一个密钥加密明文中的每个字符。所谓解密就是将明文字符和一次一用密码本上的密钥相加取模 26。

每个密钥对一条信息仅用一次。发送者加密完信息后就销毁本子上用过的一页或胶带上用过的一段密钥。接受者也有相同的密码本，用本子上相应的密钥来解密，然后销毁相同的页数或段。新的信息用新的密钥。例如，如果信息是

ONETIMEPAD

密钥为

TBFRGFARFM

那么密文是

IPKLPSFHGQ

因为 O+T mod 26= I　N+B mod 26 = P　E+F mod 26=K，等等。

假如一个窃听者无法得到用于加密信息的一次一用密码本，这种方案是最安全不过的了。一条给定的密文信息可能是任意一条同等长度的明文信息。

因为密钥字母是随机产生的，每个密钥序列均等可能，所以敌人无法知道如何破译密文。密钥序列可能是 POYYAEAAZX，那密文可能破译为 SALMONEGGS，密钥也可能 BXFGBMTMXM，密文破译为 GREENFLUID。

因为每条明文信息都均等可能，密码破译员无法决定哪条明文是正确的。一个随机密钥序列附加于非随机的明文信息，从而产生完全随机的密文信息。任何计算力量都是无法改变这个

事实。

值得注意的是密钥字母必须随机产生。任何对此方案的攻击都着眼于产生密钥字母的方法。准随机数不能算数，他们常常有非随机特性。如果用真正的随机数字源，看上去容易做起来却难得多，因为它是保密的。

另一个很重要的一点是一旦密钥序列用过一次后永远不能再用。即使你有多达几百兆的密码本，如果密码破译员有多份用相同密钥加密的密文，就可以重构出明文。将每两份密文放在一块，计算匹配的字母数目。一旦放置正确，匹配的比例会突然上升。当然确切的百分比由明文的语言而定。从这点看来，密码破译员反而更轻松。这有点像下标一致，这里比较的是两个周期而已。但这种做法只能说说，千万不能傻做。

一次一用密码本的思想可推广到二进制数据，那么二进制密码本代替了字符密码本。异或运算代替了简单的明文与密码的相加。将密码与同样的一次一用密码本进行异或即可解码。其他都不变，仍极为安全。

听起来很好，但存在着一些问题。因为二进制密钥一定要随机产生而且永远不能第二次使用，密钥序列的长度必须与信息一样。一次一用密码本可能适合很短的信息，但不适合1.544 Mbit/s 的通信信道。CD-ROM 可存储 650Mbit 随机比特序列。但问题是：①要有两份完全一样的随机比特的拷贝，而 CD-ROM 的经济价值在于复制大量而不是一样的拷贝；②要销毁用过的比特，但是除了物理地破坏整张 CD-ROM，没有设施可以擦掉其中的某部分。所以数字磁带倒是更适合的媒介。

即使解决了密钥分布和存储问题，在发信者和接受者必须保证传输中的同步。如果接受者落后一个比特或在传输中丢失一些比特，信息将毫无意义。另一方面如果在传输中一些比特很可能由于随机噪声而改变了，虽然没有添加或削减任何比特，但那几位比特就不能正确解密。这也是说明一次一用密码本没有真实性。

现在一次一用密码本主要用于超安全的低宽带信道。在美国和前苏联之间的热线谣传是用一次一用密码本加密。许多苏联间谍的情报也用一次一用密码本加密，这些情报现在还很安全，将来也是。不管超级计算机工作多长时间都无济于事。就是从仙女座（Andromeda）星球来的外星人用他们大规模宇宙飞船和无法想象的计算能力，也无法读懂一次一用密码本加密的苏联间谍情报。除非时间倒流，他们得到那个一次一用密码本。

1.6 计算机算法

诸多密码算法中最流行的三种是：

1. DES（数据加密标准），这是最流行的一种计算机加密算法，是全世界的标准。它是一个对称算法，加密和解密使用相同的密钥。

2. RSA（以其发明者 Rivest，Shamir 和 Adleman 命名），这是最流行的公开密钥算法，可用于加密和数字签名。

3. DSA（数字签名算法，作为数字签名标准的一部分），这是另一个公开密钥算法，不能用于加密，只适用于数字签名。

1.7 大　　数

本书中用了大量不同的大数来说明密码学中的问题，表 1.1 列举了一些数字的物理类似物，以免忘记这些抽象数字及含义。

表 1.1 大数

物理类似物	数 字
每天被闪电击死的可能性	1/9 000 000(2^{33})
在美国州内中奖券头奖的可能性	1/4 000 000(2^{22})
在美国州内中奖券头奖同一天又被闪电击死的可能性	$1/2^{55}$
美国每年淹死的可能性	1/59 000(2^{16})
1993 年美国因车祸而死亡可能性	1/6100(2^{13})
美国因车祸死亡可能性(按寿命)	1/88(2^{7})
到下个冰世纪的时间	14 000(2^{14})年
到太阳销毁的时间	10^{9}(2^{30})年
行星的寿命	10^{9}(2^{30})年
宇宙的寿命	10^{10}(2^{34})年
在行星中的原子数	10^{51}(2^{170})
在太阳中的原子数	10^{57}(2^{190})
在银河系中的原子数	10^{67}(2^{223})
在宇宙中的原子数(不可见除外)	10^{77}(2^{265})
宇宙的体积	10^{84}(2^{280})立方厘米
如果宇宙是封闭的:	
宇宙的寿命	10^{11}(2^{67})年
如果宇宙是开放的:	
到小质量恒星冷却时间	10^{16}(2^{61})秒
到行星脱离恒星时间	10^{15}(2^{30})年
到所有恒星离开银河的时间	10^{19}(2^{64})年
到天体轨道因向心辐射而消逝时间	10^{20}(2^{67})年
到黑洞因 Hawking 过程而消逝时间	10^{64}(2^{213})年
到所有物质冷却成零度液体的时间	10^{65}(2^{216})年
到所有物质衰变成铁的时间	$10^{10^{26}}$年
到所有物质陷落到黑洞的时间	$10^{10^{76}}$年

这些数是以数量级表示的估计量,来源于各种地方。Freemen Dyson 在其论文“无端的时间,在开放宇宙中的物理学和生物学”中解释了许多天体物理学数字。汽车事故死亡数字由交通统计部门的 1993 年每百万人 163 起交通事故得到,人的平均寿命为 69.7 岁。

第一部分 密码协议

第2章 组合协议

2.1 协议简介

密码学总的要点是问题求解(实际上,这也是计算机的要点——有时许多人易于忽略)。密码学要解决的问题围绕着安全和隐私以及防范不可信和不诚实的人。算法和技术的学习,只有为了解决问题,才能引人入胜。于是我们首先来讨论协议。

协议是设计来完成一个任务的一系列步骤,密码学的协议牵涉双方或多方的关系。这是一个重要的定义。"一系列步骤"意味着协议有一个过程,从开始到结束。每一步骤必须依次执行,在前一步骤完成之前,后一步骤是不能执行的。"牵涉双方或多方"意味着至少需要两个人完成协议,一个人单独不能完成协议。当然,一个人能执行一系列步骤完成一个任务(如烘面包),但这不是一个密码学协议。最后,"设计来完成一个任务"意味着协议必须做一些事。看似像一个协议但不能完成任务的事不是协议,纯粹是浪费时间。

协议的其他特性:

1. 与协议有关的任何人必须知道协议和事先安排的所有步骤。

2. 与协议有关的任何人必须同意按照协议执行。

3. 协议必须是不含糊的,每一步必须明确地定义,必须正确地理解。

4. 协议必须是完整的,对每一个可能的解决方法必须有一个专门的动作。

本书中的协议由一系列步骤组成。协议的执行为线性地从头到尾地按步处理,除非有指令转到另一个步骤的分支。每一步骤至少包含下列事项之一:一方或多方的计算,或一方至他方的信息。

密码协议

密码协议是使用于密码学的协议。有关的多方可能是朋友,可绝对信任的另一方,或是对手和不可信任的另一方。当然,密码协议涉及到一些密码算法,但一般地说,协议的目标并不单单是安全。参与协议的一方可能想共享部分安全去作计算,共同产生随机序列,为对方确认他们的身份,或同时签发合同。完成这些目标的密码协议已完全改变了我们的初衷,相互不信任的双方能通过网络完成协议。

协议的目的

在日常生活中,几乎每一件事都有非正式的协议,电话购物,玩扑克,选举投票。没有一个人去想及这是协议,随时间推移每个人都知道如何去做并得成功。

越来越多的人通过计算机网络替代面对面的通信。人们容易适应新情况,计算机几乎没有这种灵活性。计算机需要正式的协议来做人们未曾考虑的事情。

许多面对面的协议依靠人的态度来确定公正和安全。给一个陌生人一叠钱去买东西;看不见他们的洗牌和出牌,同一些人玩扑克;给政府投票而不需用匿名方式等都靠面对面的了解。

假设计算机网络上的人仍是天真的,诚实的;假设计算机网络管理者是天真的,诚实的。甚

至假设计算机网络的设计者是天真的，诚实的。虽然大部分人是这样，但我们需要防止极少部分不诚实的人。形成协议，我们能以这种方式检查欺骗和挫败欺骗者。

除规范之外，协议还用来从完成任务的机理中抽象完成任务的过程。两台计算机之间的通信，不论计算机是 IBM－PC，VAX 或传真机对通信协议来讲是相同的。这种抽象允许我们检查协议特性，而不陷入完成细节。当我们确信有一个好的协议时，我们能在从计算机到电话的任何设备上实施它。

为了说明协议，将有关人物列表表示。(见表 2.1)A 和 B 是第一、二个人，他们将执行所有通常二个人的协议。作为规则，A 将初始化所有协议。B 将完成其次部分。如果协议需要第三个或第四个人，他们是 C 和 D，其他角色各守其职。

表 2.1 人 物 表

A	参与所有协议
B	参与二、三、四方协议
C	参与第三、四方协议
D	参与第四方协议
E	偷听者
M	蓄意攻击者
T	可信任的仲裁人
W	管理员：在一些协议中将保护 A 和 B
P	证明人
V	验证者

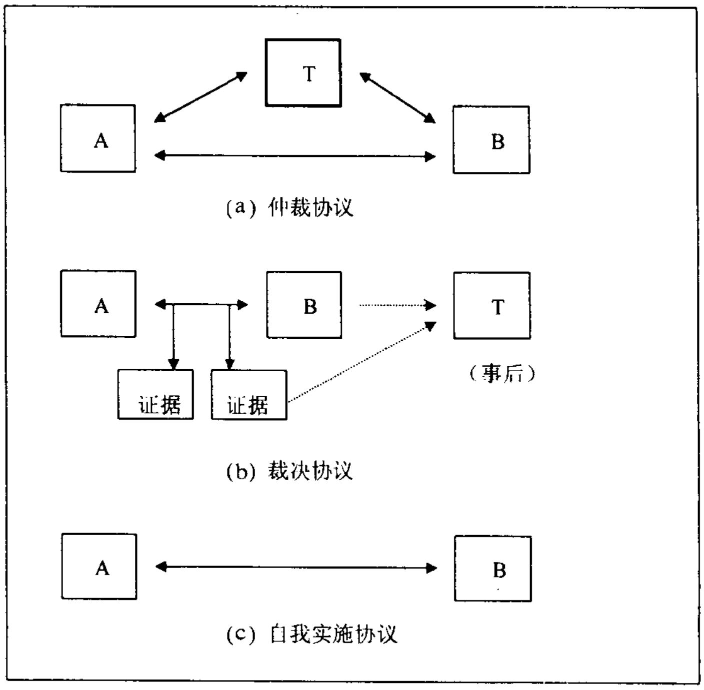


图 2.1 协议的类型

仲裁的协议

仲裁者是一个没有利害冲突并可完全信任的第三方，完成第一部分协议(见图 2.1a)。没有利害冲突意味着不参与和卷入协议。可信指协议有关人员承认他说的是真的，做的是正确的，他的协议部分也是完整的。仲裁者能帮助两个相互不信任的双方完成协议。

在现实世界中，律师经常扮作仲裁者。例如，A 卖一辆汽车给一个陌生人 B。B 想支付支票，但 A 无法知道支票真假。A 在把产权交给 B 之前，她想核对清楚。同样 B 也不信任 A，在没有得到产权之前，他不想交出支票。

找到一个两者信任的律师。借助于律师，A 和 B 能同意下列的协议，以确保不相互欺骗。

(1)A 把产权和钥匙交给律师；

(2)B 把支票交给 A；

(3)A 抵押支票；

(4)在等待验证支票真伪的一段特定时间后，律师把产权交给 B；

(5)如果在特定的时间内验证支票为假，A 把这个证据给律师看，律师把产权和钥匙还给 A。

在这个协议中，A 相信律师不会在支票判明为真之前把产权和钥匙交给 B，如果支票为假，把东西返回给她。在律师把产权和钥匙交给 B 之前，B 相信律师会保留它们直到支票为真。

律师不关心支票是否真假，仅做按照各种情况他该做的事。

在这个例子中，律师扮演一个契据代理人。契据代理人经常仲裁金融交易。律师有时做为遗嘱和合同转让的仲裁者。或股票买、卖双方的仲裁者。

银行家也仲裁协议。B 能使用一张有效支票从 A 那里买一辆汽车：

(1)B 开一张支票，交给银行；

(2)在 B 存入足够保证支票的金额后，银行签支票并交给 B；

(3)A 把产权和钥匙交给 B；

(4)B 把签证支票交给 A；

(5)A 提取支票。

因为 A 信任银行家，所以协议可操作。

公证人是另一种仲裁人。当 B 从 A 收到一份有公证的文件时，他相信 A 亲笔签名的文件和她的笔迹。如果需要的话，公证人能到法院，证明事实。

仲裁者的概念像社会一样古老。总是有人，如统治者、牧师等等，担当公证行为。仲裁者在社会中有一定的任务和职位，背判公众的信任将使社会受危害。于是玩弄契据的律师肯定要被取消律师资格。现实世界并非永远是玫瑰，维护正义是人们的理想。

这种理想能转入计算机世界，但计算机中的仲裁者面临以下问题：

· 如果你知道他是谁，能看见他的脸，或感觉他是一个诚实的人，则容易信任这中立的第三方。在网络上互相怀疑的双方也同样怀疑不见面的仲裁者。

· 计算机网络必须承受维持仲裁的成本。我们都知道律师要收费，可谁来承担网络的总开销？

· 任何仲裁协议都有一段时延。仲裁者同时处理各业务。

· 任何协议的大规模实施中仲裁者是潜在的瓶颈。增加仲裁者的数量能解决这个问题，但成本增加。

· 因为网络上的每一个人必须相信仲裁者，他成了任何试图破坏网络人的攻击点。

裁决协议

因为雇用仲裁者的高成本，仲裁协议能划分为两个低层子协议。一个为非仲裁子协议，每次人们想完成协议时执行。另一部分是仲裁子协议，当有争议时，仅在本环境中执行。这个特殊的仲裁者称为裁决者(见图 2.1b)。

裁决者也是一个无偏见的值得信任的第三方。与仲裁者不同，他不直接卷入每一个协议。裁决者仅来决定交易完成得是否公正。

法官是一个职业裁决者。与证人不同，他仅在发生争论时引入。A 和 B 签署合同可不要法官，除非他们中之一告到法庭，法官才会看他们的合同。

这个签署合同的协议形成方式为：

非仲裁子协议(每次执行)：

(1)A 和 B 协商合同的用语；

(2)A 签合同；

(3)B 签合同。

裁决子协议(仅在争论的情况下执行)：

(1)A 和 B 出现在法官面前；

(2)A 递交他的证据；

(3)B 递交他的证据；

(4)法官判断证据。

裁决者和仲裁者之间的关键差异为裁决者并非总是必需的。只在有争议时叫法官判断，如果没有争议，不需法官。

有一些带裁决的计算机协议。这些协议相信有关方是诚实的。但如果有人进行欺骗活动，通过收集到的数据，使得无偏见的第三方判定是否有人进行欺骗活动。在一个好的裁决协议中，裁决者也能决定欺骗者的身份。在现实生活中，几乎少有裁决者。必要的检查防止了欺骗活动，使人们保持诚实。

自行实施协议

自行实施协议是最好的一类协议。协议本身保证公正(见图 2.1c)。不需要仲裁者完成协议。不需要裁决者解决争议。构成的协议没有任何争议。如果一方企图进行欺骗，另一方立即检测出并停止协议。无论欺骗方希望什么，都无法实现欺骗行为。

各种世界模式的最佳情况是，每个协议将自行实施。不幸，每种情况下都没有自行实施协议。只能尽力而为罢了。

对协议的攻击

对协议的攻击能直接作用到用于协议的密码算法，用于实现算法的密码技术或者协议本身。本节仅讨论对协议本身的攻击，假定密码算法和技术是安全的。

有各种类型的人试图攻击协议。与协议无关的某些人能窃听部分或全部协议。这称为被动攻击。因为攻击者不影响协议。他所能做的是观察协议和试图获得信息。这种攻击相应于唯密文攻击。

另一方面，攻击者试图按他的利益改变协议。他在协议中引入新的信息，删除存在的信息，替代一部分信息，破坏通信信道，或改变存在计算机中的信息。这些称为主动攻击，因为他们需要主动地干涉系统。

被动攻击者仅获得与协议有关方的信息。他们做这些为了收集通过各方的信息，试图分析他们。换句话说，主动攻击者有许多特殊的目的。攻击者对获得信息、降低系统性能、破坏存在的信息、或得到非权限访问的资源感兴趣。

主动攻击者是非常严重的，特别在不同的双方相互信任的协议中。攻击者并非是完全的局外者，他们将是一个合法的系统用户。甚至有许多主动攻击者，一起工作，每一个都是合法的系统用户。

攻击者也有可能是协议有关的一方。他们可能在协议期间说谎，并不遵守协议。这类攻击者称为欺骗者。被动欺骗者遵守协议但试图获得比协议允许的更多的信息。主动欺骗者破坏进行中的协议，试图进行欺骗。

如果大部分相关的用户是主动欺骗者，要维持系统的安全是非常困难的，但有时，合法的用户可能检测出主动欺骗者在进行欺骗。当然，协议应当安全地防止被动欺骗。

2.2 对称密码通信

双方怎样安全地通信？将通信加密。完成一个协议往往比协议本身更复杂。让我们看看，A 发一份加密信息给 B 将会发生什么？

(1)A 和 B 同意一个密码系统；

(2)A 和 B 同意一个密钥；

(3)A 取他的明文，用加密算法和密钥加密明文信息，产生一个密文信息；

(4)A 发密文给 B；

(5)B 用相同的算法和密钥解密密文信息然后阅读。

A 和 B 之间的一个窃听者 E，从这个协议过程中听到什么？如果他只窃听到步骤(4)的传输，他必须试图对密文进行密码分析。这个被动攻击为仅知密文攻击；有一组算法可因窃听者计算能力的局限而阻止其企图。

但是如果 E 知道步骤(1)和(2)，他就成功了。知道算法和密钥，当信息按步骤(4)通过通信信道时，E 与 B 一样仅需解密而已。

这就是为什么密钥管理在密码学中这样重要。好的加密系统其安全性寓于密钥而不在于算法。用对称算法，A 和 B 公开完成步骤(1)，但必须秘密地完成步骤(2)。协议前后及执行当中，密钥必须保持安全；否则，信息将不安全(公钥密码术用另一种方式解决)。

更有甚者，攻击者 M 试图破译步骤(4)的通信信道，使 A 几乎不能同 B 通信。M 也截获 A 的信息，如果他知道密钥(截获步骤(2)的通信，或破译密码系统)，然后加密自己的信息，替代截获的信息发送给 B。B 无法知道信息是否来自 A。如果 M 不知道密钥，他仅能产生无用的替换信息。B 接到后，能推出网络或 A 有一些严重的问题。

再看看 A，她能做些什么破坏协议的事。她能复制一个密钥给 E。现在，E 能阅读 B 的任何信息。B 不知道 E 有密钥，认为安全地同 A 通话。他不知道 E 正将他的话发布给传媒。这也是严重的，但协议没有问题。无法停止 A 在协议中复制明文给 E。当然，B 也能同样做。假设 A 和 B 相互信任是该协议成功的前提。

小结：对称密码系统有下列问题：

· 如果密钥泄密(偷窃，猜测，敲诈，贿赂等)，那么有密钥的对手能解密用那个密钥加密的所有信息流。也能假装为一方，产生一些错误信息来愚弄另一方。经常改变密钥减少这类失误是非常重要的。

· 密钥必须安全地发送。他们比任何加密的信息更易受攻击，因为知道密钥意味着知道所有信息。对跨接世界的加密系统，这是一个艰难的任务。经常由信使带密钥到目的地。

· 假设网络上的每一对用户使用一个独立的密钥，密钥的总数随用户数而迅速地增加。例如，10 个用户需要 45 个不同的密钥相互谈话；11 个用户需要 55 个密钥。保持小用户数能减少麻烦，但那是不现实的。

2.3 单向函数

单向函数的表示是公开密钥密码学的中心。尽管它本身不是协议，但单向函数是本书描述的大部分协议的基石。

单向函数是一个相对容易计算的函数，但逆运算却相当困难。即，已知 X，容易计算 f(X)，但已知 f(X)，难以计算 X。“难”意味着即使世界上所有的计算机都处理这个问题也要花几百万年时间。

单向函数的一个浅显的例子就是拆开手表。容易把手表拆成几百个细小的小部件。然而，把所有的那些细小的部件放在一起成为手表是不容易的。

但事实上它还不是真正的单向函数。严格地说，无法证明单向函数存在，或没有任何能构造他们的实际证据。许多函数看似单向的，我们能有效地计算他们，只是没有容易的方法作逆运算。例如，X^2 容易计算，而对 X 求根就比较难。但我们仍假定单向函数存在。

那么，单向函数好在哪里呢？他们不能用来进行加密。用单向函数加密是没有用的；没有一个人能解密。试想，在板上写信息，把板打碎成各含一些比特信息的小块，把它们交给朋友，要求他读信息，观察你的朋友怎样引用单向函数。为了加密，我们需要有机关的单向函数。

机关单向函数是特殊种类的具有秘密机关的单向函数。一个方向容易计算，另一个方向难以计算。但是，如果你知道机关，你能容易地作逆运算。即，已知 X 容易计算 f(X)，已知 f(X)难以计算 X。然而，有一些秘密信息 Y，使得已知 f(X)和 Y 容易计算 X。在手表的例子中，秘密信息可能是手表的元件装配图。

邮箱是一个机关单向函数的例子。任何人容易把邮件放入信箱；信箱上仅有一条缝和一个门。把邮件放入邮箱是公开的活动。打开信箱并不是公开的，很难；你需焊枪或其它工具。然而，你有秘密机关（钥匙或密码锁的密码）就容易打开信箱。公开密钥密码学非常类似如此。

2.4 单向哈希函数

单向哈希函数有许多名称：压缩函数、抽象函数、信息摘要、指纹、密码校验和、数据综合校验(DIO)、施行检测码(MDC)、信息认证码(DAC)。无论你如何称呼，它是现代密码学的中心。单向哈希函数是许多协议的另一类构造模块。

单向哈希函数在计算机科学中使用很长时间了。哈希函数是数学的或其他方面的一个函数，它将输入字符串转换为固定大小（经常为更小）的字符串输出。简单哈希函数取决于输入字符串，它将所有输入字符 XOR，组成一个字节后输出。这里，要点是识别输入字符串。产生的值能指出候选输入字符串。因为哈希函数是典型的多对一函数，我们不能使用他们肯定二个字符串相同，但我们能使用他们得到相同性的合理的保证。

单向哈希函数容易从输入字符串计算哈希值，但难以产生一个字符串使它做哈希运算后达到某一确定值。上文中的哈希函数不是单向的，给出特定的字节内容后，去找一个字符串使其异或后为该字节是很平常的。

实际的单向哈希函数可以返回的值为 128 比特长，使得有 2^{128}可能的哈希结果。需要找出一个具有相同哈希值的随机字符串的试验量，为 2^{128}，需要找出两个有相同（随机）哈希值的试验数目是 2^{64}。

有两个主要类型的单向哈希函数：具有密钥和无密钥。无密钥的单向哈希函数能被任何人计算，是单向的输入字符串的函数。具有密钥的单向哈希函数是输入字符串和密钥的函数；具有密钥的人能计算哈希值（同单向哈希函数计算相同，然后加密）。

已开发了专门为单向哈希函数设计的算法。这些是伪随机函数；任何哈希值大致机会均等。输出不取决于任何可察觉的输入。一般而讲，输入任一比特的单独变化会影响输出的半数比特。已知一个哈希值，找出哈希值对应的输入字符串是不可实现的。

试想把它们作为指纹文件之用，如果你想检查某些人有一给定文件（你也保存这一文件）又不要让对方再发送给你，则只须问其哈希值；如果他发给你一个正确的哈希值，那么他有那个文件几乎是肯定的。通常，使用无密钥的单向哈希函数时可使任何人都能进行验证。如果仅想使指定的接收者能验证，那么使用带密钥的单向哈希函数。

2.5 公钥通信

可以把对称算法想象成保险箱，密钥是密码。具有密码的人能打开保险箱，把文件放入其中，再关闭它。没有密码只能撬开保险箱。

1976 年 Whitfiled Diffie 和 Martin Hellman 改变了密码学从来的惯例，提出了公开密钥密码学，用两个不同的密钥取代一个密钥，一个公开的和一个秘密的。此外，从公钥不可能推知私钥。有公钥的任何人能加密信息，但不能解密。仅有私钥的人能解密信息。就像一些人在密码箱上开一个狭缝，任何人能把信息从缝中塞入，但仅有私钥的人能打开保险箱去阅读。

数学上，用上述机关单向函数进行处理。加密是容易的。指导加密的是公钥，任何人能加密信息。反之解密则是十分困难。如果没有私钥，难到足以用 Gray 巨型计算机化几千年(百万年)时间不能解密。有了私钥，解密就像加密一样容易。

以下是 A 怎样使用公钥密码学发信息给 B：

(1)A 和 B 同意公钥密码系统；

(2)B 发给 A 他的公钥；

(3)A 使用 B 的公钥加密她的信息并发给 B；

(4)然后 B 使用他的私钥解密 A 的信息。

注意公钥密码术怎样解决对称密码系统的主要问题：分发布密钥。在传统的密码系统中，A 和 B 必须同意用相同的密钥。A 随机选择一个，但必须事先交给 B。她用挂号邮件发给他，但花时间。用公开密钥密码术，就没有问题了，无须事先协调。A 发一个秘密信息给 B。窃听者听到全部交换过程，包括 B 的公钥和用密钥加密的信息，但窃听者无法得到 B 的私钥，也就不能恢复密文。

一般地讲，网上用户采用公钥密码系统。每一个用户有他或她自己的公钥和私钥，公钥全部公布在某数据库中。协议更简单了：

(1)A 从数据库中取 B 的公钥；

(2)A 用 B 的公钥加密她的信息并发给 B；

(3)B 用他的私钥解密 A 的信息。

在第一个协议中。在 A 发给 B 信息之前，她必须发送她的公钥给他。第二个协议更像传统的邮件。直到 B 想读她的信息，才与协议有关。

对公钥密码术的攻击

在所有公钥密码术协议中，避开了 A 怎样得到 B 的公钥的问题。

交换公钥的易行方法是从某地安全的数据库中存取。数据库必须是公开的，使得任何人也能得到公钥。数据库必须防止可信任者之外的任何人的访问；否则，攻击者将用他选择的密钥替代 B 的。他那样做后，B 再也不能读到给他的信息，所有给 B 的信息都被攻击者取得。

即使公钥存放在安全的数据库中，攻击者在传送过程中仍然能替换。为防止这种情况，仲裁人用他自己的私钥为每一个公钥签名。当使用这种管理时，经常称仲裁人为密钥验证管理机构或密钥分布中心(KDC)。在实际实施中，KDC 签署一个合成的信息，由用户的名称、公钥和关于用户的任何重要信息组成。这个签名的合成信息存储在 KDC 的数据库中。当 A 得到 B 的密钥时，她验证 KDC 的签名确保她自己取得有效密钥。

这样之后，攻击者也并非不可能攻击，仅仅是更困难。A 仍有存储在某地的 KDC 公开密钥。攻击者不得不用他自己的公钥去替换，破坏数据库，用他自己的密钥替代有效的密钥，就像他是 KDC，煞有介事地用他的私钥签名。如果连纸上的亲笔签名也可被伪造的话，攻击者引起的麻烦就大了。

混合密码系统

公钥算法比对称算法慢；对称算法至少比公钥算法快 1000 倍。在大部分实际实施中，公钥

密码术被用在会话密钥的保安和发送中，那些会话密钥被用于对称算法以确保信息流安全。这就称为混合密码系统。

2.6 数字签名

在文件上手书签名已经长期被用为原作者的证明，或用来表示同意文件所列的条款。签名不可辩驳的事实是：

1. 签名是不可伪造的，签名是签名者慎重签在文件上的证明。

2. 签名是可信的。签名使文件的接受者相信文件已由签名者慎重签名。

3. 签名是不可再用的。签名是文件的一部分，不择手段的人不能把签名移到不同的文件上。

4. 签名文件是不可能改变的。文件签名后，不能改变。

5. 签名不能否认，签名和文件是一个物理事件，签名者以后不能宣布他或她没签名。

在实际中，上述叙述没有一个完全是真的。签名能被伪造，签名能从一张纸上剽窃来并移动另一处，签名后文件能被改变。然而，因为欺骗的困难和检测的出现，所以我们仍愿与这些问题一同生活。

要想在计算机上签名，有很多问题。首先，比特流容易拷贝。即使人的签名是难以伪造(例如手写，签名的图形图像)，但把一个文件的有效签名移动到另一个文件中是容易的。这种签名毫无意义。第二，签名后，仍容易修改文件，并不留下任何修改的痕迹。

用对称密码系统和仲裁人的签名文件

A 想在数字信息中签名并送给 B。她能在仲裁人 T 和对称密码系统的帮助下实现，一个有权力的、可信任的仲裁人能同 A 和 B 通信(或想要用在数字文件签名的任何人)。他享有一个对 A 的私钥 K_A，它不同于 B 的私钥。在协议开始之前，这些密钥已经建立很久了，能对多个签名重复使用。

(1)A 用 K_A 加密她给 B 的信息，发给仲裁人；

(2)仲裁人用 K_A 解密这些信息；

(3)仲裁人取出解密信息，叙述他从 A 收到的信息，拷贝 A 的加密信息。他用 K_B 加密全组信息；

(4)仲裁人发加密信息组给 B；

(5)B 用 K_B 解密组信息。他能读信息和 A 送给仲裁人的证书。

B 怎样知道信息来自 A，而不是来自某些骗子？这是从信息的加密中推断出的。因为仅仅他和 A 共享他们的私钥，A 只使用它加密信息。

这与纸上签名一样吗？让我们看看期望之点：

1. 签名是不可伪造的，仅仅 A 知道 K_A，使得只有 A 能用 K_A 加密信息并发给仲裁人。如果有人假冒 A，仲裁人立刻在步骤(2)识别这种情况，不发送信息给 B。

2. 签名是可靠的。仲裁人知道信息来自 A。仲裁人的验证是充分的证明。

3. 这个签名是不可再用的。如果 B 试图取出仲裁人的验证并附属于另一个信息上，A 大叫违规了。仲裁人要求 B 生成信息和 A 的加密信息。然后，仲裁人用 K_A 加密信息，注视是否与 B 交给他的加密信息相匹配。当然，因为他不知道 K_A，B 不能生成匹配的加密信息。

4. 签名文件是不可改变的。B 收到后，试图改变文件，仲裁人将防止如上述所用一样的不正当手段。

5. 签名文件不得否认。即使后来 A 宣布她从没发送信息。仲裁人的验证会作证。每个人都信任仲裁人说的是真的。

如果 B 想给 C 显示 A 签名的文件，他不会对她展示他的私钥。他必须再经过仲裁人：

(1)B 取出信息并叙述信息来自 A，用 K_B 加密他们，发送他们给仲裁人。

(2)仲裁人用 K_B 解密组信息。

(3)仲裁人用他与 C 共享的 K_C 私钥再加密组信息，发送给 C。

(4)C 用 K_C 解密组信息。现在他能读信息和 A 发送的仲裁人的验证信息。

这些协议工作时，对仲裁人来说很费时间。他必须花费时间解密和加密，就像每对想发签名文件给另一人之间的中间者。他将是任何通信系统的瓶领，即使他是一个软件程序。

制造和维持仲裁人是较困难的，网络上的每个人都信任他。仲裁人必须确实可靠；即使他在百万次签名中产生一次错误，就会没有人继续信任他。仲裁人也必须完全是安全的。如果他的秘密密钥数据库泄露，或一些人设法修改他的代码，每个人的签名将完全无用了。结果是一片混乱。以上只是理性的考虑，实际工作并非这样完美。

用公钥密码术的签名文件

公钥算法能被用于数字签名。在一些算法中公钥或私钥能被用来加密。使用你的私钥加密文件，你有一个秘密的数字签名。在另一种情况一个独立的算法能用数字签名而不能用来加密。这个想法首先由 Diffie 和 Hellman 发明，随后被其他人进一步发展和详细论述。

基本协议是简单的：

(1)A 使用有其私钥的数字签名算法在信息上签名；

(2)A 发文件给 B；

(3)B 使用具有 A 的公钥的数字签名算法来验证签名。

这个协议比前面的协议好，并不需要仲裁。A 和 B 他们自己能判断。B 甚至不需要仲裁人来解决争端，如果他不能完成步骤(3)，立即知道签名是无效的。

这个协议也满足我们期望的特性：

1. 签名是不可伪造的。仅 A 知道她自己的私钥。

2. 签名是可靠的。当 B 验证了带 A 公钥的信息时，他知道 A 签名的信息。

3. 签名是不可再用的。签名是文件的函数，不能转到另一个文件上。

4. 签名文件是不可改变的。如果文件有任何变动，不能被 A 的公钥验证。

5. 签名不能否认。B 不需要 A 帮助来验证她的签名。

签名文件和时戳

实际上，B 在某些情况下能欺骗 A。他能把签名和文件合在一起再次使用。如果 A 签了一份合同，还不怎么样，但如果 A 签一张数字支票，这将是严重的。

如 A 发给 B 一张 100 元的签名数字支票。B 带支票到银行验证签名，把钱从一个帐号移到另一个帐号。假如 B 为一个不择手段的人，保留一张数字支票的复制品。几周后他再次拿到银行(或可能为不同的银行)。银行验证签名，把钱从一个帐号移到另一个帐号。如果 A 不结算她的支票本，B 能永远这样做。

于是，数字签名经常包含时戳。把签名的日期和时间附属在信息上，签名作为信息的其余部分。银行将这个时戳存储在数据库中。现在，当 B 试图第二次兑换 A 支票时，银行从数据库对照检查时戳。因为银行已经为 A 兑换了一张相同时戳的支票，银行报警而使 B 在监狱中度过 15 年。这时他也许会去学法律或密码协议。

用公钥密码术和单向哈希函数签名文件

在实际实施中，公钥经常不是有效的，乃至不能加密长的文件。为节省时间，数字签名协议经常用单向哈希函数实施。A 用对文件哈希值的签名，来代替文件签名。在这个协议中，单向哈希函数和数字签名算法事先已取得一致。

(1)A 生成文件的单向哈希值。

(2)A 用私钥对哈希值签名，也就是对文件签了名。

(3)A 发送文件和签名哈希值给 B。

(4)B 生成 A 发送的文件的单向哈希值。然后验证 A 的公钥签名的哈希值，如果匹配，签名有效。

因为同一 160bit 哈希值的两个不同文件的变化仅仅是 2^{160} 分之一，任何人能安全地认为哈希签名与文件签名等同。如果不使用单向哈希函数，由相同哈希值来产生多重文件是容易的事，使得可用任何签名来伪造其他文件上的签名。所以不用单向哈希函数，这个协议无法实现。

这个协议的优点是，首先，签名免于从文件中分离。其次，文件和签名的接收存储量很小。

档案系统能使用这类协议来验证文件的存在而不存储他们的内容。中心数据库只有文件的哈希值，几乎不必看到文件。用户向数据库提交他们的哈希值，数据库打上时戳并作存储。如果在将来有人对由谁和何时生成文件有异议时，将从数据库中找出文件中的哈希值解决这个问题。这对保密是非常有意义的；A 可取得版权证明但仍不暴露文件。

算法和术语

有许多数字签名算法。他们都是公开密钥算法：包括用一些秘密信息给拥有者对文件签名和用一些公开信息允许任何人来验证文件。有时，把签名过程称为用私钥加密，而把验证过程称为用公钥解密。这是一种误导，因为只有一种算法是这样的。另外，实施方法也有差异。例如，单向哈希函数和时戳有时对签名和验证过程增加额外的步骤。许多算法能用于数字签名，但不能用于加密。

一般地说，如不涉及算法的细节，用私钥 K 签名表示为：

$S_K(M)$

用相应的公钥验证表示为：

$V_K(M)$

签名后加到文件上的位串(上例中，用私钥加密的单向哈希值)称为数字签名。信息的接收者相信了发送者的身份和全部信息的整个协议，称为验证。

多重签名

A 和 B 怎样签同一数字文件？不用单向哈希函数可有两个选择。一是 A 和 B 分别签文件的复制品，产生两份原始文件，这不理想。二是 A 首先签文件，然后 B 签 A 签过的文件。仅当只验证 B 的签字而不可能验证 A 的签字时，这样做是行得通的。

如果在文件的单向哈希值上签名替代文件签名本身，多重签名是容易实现的：

(1)A 签文件；

(2)B 签文件；

(3)B 发送他的签名给 A；

(4)A 或 B 把文件、他的签名或她的签名给 C；

(5)C 验证 A 的签名和 B 的签名。

A 和 B 能并行或串行执行步骤(1)和(2)。在步骤(5)C 能验证一个签名而不必以验证其他

为先决条件。

数字签名上的欺诈

A能在数字签名上进行欺诈，她自愿签署了文件，以后却宣布没有签过。为此，首先，她正常地签文件；然后，她匿名地公开她的私钥，进而并在公开场合宣布原密钥作废，于是，得知该密钥的任何人都能自称是A并签署文件。A宣布她的签名已被泄露，有人正在使用它，于是她不承认签名文件和使用私钥签署的任何东西。时戳能限制这种欺骗活动，但A总是能把私钥泄露期提得更早，如果A很好地安排时间，她能签署文件并成功地掩盖为不是她签的。

虽然还没有什么办法对付这种侵犯，人们能采取步骤保护旧的签名，不因发生新的争执而失效。例如根据仲裁人的时戳仅仅是报告丢失以后的签名文件被认为无效。这类似于报失信用卡的规则一样。

数字签名的应用

数字签名的最早应用建议是使核试验禁止条约的监视变得容易。美国和前苏联把地震测试仪放在相互的国土上监视核试验。问题是监视国必须得到保证东道国不干扰其地震仪的数据。同时，东道国也要得到保证监视仪仅发送需要监视的专门信息。传统的验证技术只能解决第一个问题，但数据签名把两个问题都解决了。东道国能读但不能改变来自地震仪的数据，监视国能知道数据是否被篡改。

2.7 加密数字签名

数据签名与公钥密码术相结合，发展了一个把加密的安全性与数字签名的认证相结合的协议。就像在信封上签名一样：数字签名提供对原作者的证明，加密提供隐私保护。

(1)A用她的私钥对信息签名。

$S_A(M)$

(2)A用B的公钥加密签了名的信息并发给B。

$E_B(S_A(M))$

(3)B用他的私钥解密信息。

$D_B(E_B(S_B(M)))=S_A(M)$

(4)B用A的公钥进行验证并恢复信息。

$V_A(S_A(M))=M$

当然，协议使用时戳能防止信息的再使用。时戳也能防止其他潜在的圈套。

读取确认接收的信息

对确认附加特性的协议实施。无论何时收到信息，都返回给发送者一个确认的收据。

(1)A用她的私钥签署信息，用B的公钥加密，并发给B。

$E_B(S_A(M))$

(2)B用他的私钥解密信息，用A的公钥验证签名，从而验证A签名的信息和恢复信息。

$V_A(D_B(E_B(S_A(M))))=M$

(3)B用他的私钥签署信息，用A的公钥加密，并返回给A。

$E_A(S_B(M))$

(4)A用她的私钥解密，用B的公钥验证签名。若最终的信息与她发送给B的相同，她知道B正确地收到了信息。

如果相同的算法用于加密和数字签名，有一种可能的攻击会发生。在这些情况，数字签名

操作与加密操作相逆：$V_X=E_X$ 和 $S_X=D_X$。

假设攻击者 M 是一个合法的系统用户，具有他自己的公钥和私钥。现在，让我们观察他偷阅 B 的邮件的情况。首先，他记录在步骤(1)A 给 B 的信息。过一会，他发一个信息给 B，声称它来自他一个合法用户(但实际是一个攻击者)。B 认为这是合法信息，用他的私钥解密信息并试图验证签名。最终是一些莫名其妙的结果。

$$E_M(D_B(E_B(D_A(M))))=E_M(D_A(M))$$

尽管这样，B 继续遵循这个协议并给 M 发回回执：

$$E_M(D_B(E_M(D_A(M))))$$

M 用他的私钥解密回执信息，用 B 的公钥加密，然后用他的私钥再次解密，再用 A 的公钥加密。瞧！攻击者得到了 A 发给 B 的信息 M。

设想 B 会自动地发给攻击者一个回执并不是不合理的，例如这个协议可能是嵌在通信软件中，会自动地发送一个回执给接收者。尽管产生的回执信息莫名其妙，并带来了不安全但还是自觉自愿地执行了。如果 B 在发送回执之前，检查信息是否能理解，他将避免这种不安全的问题。

允许攻击者给 B 发送他窃听来的不同信息会加强这种攻击。重要的安全措施是决不要去签署来自其它人的任意信息或加密任意的信息并把结果交给其他人。

破除重发攻击

由于加密操作与签名验证操作相同，解密操作与签名操作相同，上述攻击得以实现。如果协议使加密和数字签名稍有不同，这个问题就可避免。单向哈希函数数字签名能解决这个问题；因为每一个操作使用不同的密钥；同样在打时戳上也使进入信息和出去的信息时戳不同。

一般地说，这个协议是相当安全的：

(1)A 签署信息；

(2)A 用 B 的公钥加密信息和签名(使用的加密算法与她用来签名的算法不同)并发给 B；

(3)B 用他的私钥解密信息；

(4)B 验证 A 的签名。

协议还可改善，使得 A 在签名前加密信息，从而适合中介方需要验证签名而不必读信息的情况，一般而言，这是较好的加密方式。几乎不给窃听者任何孔子可钻。

2.8 随机和伪随机序列生成

为什么要专为密码随机数生成而大伤脑筋呢?在许多编译程序中都已有了随机数生成器，只要去调用就可以了！不幸，那些随机数生成器几乎在密码学上都不安全，甚至不可能是完全随机的。他们中的大部分令人不敢恭维。

在大部分简单应用中，随机序列生成器不必具有严格的随机性。像计算机游戏只需少数不显眼的随机数。然而，密码术对随机数的特性非常敏感。使用性能差的随机序列生成器，会得到不可思议的相互关系和奇怪的结果，从而破坏安全。

软件实现的随机数生成器不能产生随机序列，其结果甚至不像随机序列。因为，计算机不可能产生真正的随机事件。Knuth 引用了 Von Neumanm 所说的："任何人认为可以用算术方法产生随机数字是违背常规的"。计算机是确定的从一端输入，经过内部完全可预测的操作，不同的结果从另一端出来。无论是不同的情况还是用不同的计算机出现的结果相同。

计算机仅存在有限的状态，出来的结果总是输入和计算机目前状态的确定函数。因而任何

计算机上的随机序列生成器是确定的、周期的、可预测的，而不是随机的。真正随机序列生成器需要一些随机输入，但计算机不能提供。

伪随机序列

最好的计算机能生成一个伪随机序列生成器。序列的周期将长达至合理的限度以至于实际使用时不感到它的周期性。应用短的非周期的子序列难以将它和真正的随机子序列作出区分。随机特性可实验性地测试，并用 Chi 平方测试的统计表达式进行比较。

为此，如果有以下特性，序列生成器为伪随机序列：

看似随机，意为它能通过给出的所有随机统计测试

在计算机上产生好的伪随机序列的工作仍在继续。在学术文献中有大量发生器和各种随机测试方法的报道。它们都是周期性的（无法避免）；周期可能为 2^{256} 和更长，以满足高要求的应用场合。

问题仍是那些不可思议的相互关系和奇怪的结果。如果你在某种方式下使用时，每一个有限发生器产生的这类结果会引来密码破译者的攻击。

密码安全的伪随机序列

伪随机序列生成器在密码术中应用得比其它应用场合更多。密码学的随机性并不仅意味着统计的随机性，它必须附加第二个持性，即它是不可预测的。给定完整的算法知识或硬件随机序列生成器和已有的比特流，下一随机位从计算上必须是难以预测的。

像任何密码算法一样，安全的伪随机序列生成器也是攻击目标。就象可能破译加密算法一样，破译密码安全的伪随机数生成器也是可能的。

实际的随机序列

从哲理上考虑，随机事件存在吗？随机序列是什么？怎样知道序列是否为随机的？“101110”比“1010101”更随机吗？量子力学告诉我们在现实世界上有货真价实的随机性。但是回到计算机芯片和有限状态机的宏观世界时，随机性能维持吗？

撇开哲理，如果序列生成器附有第三个特性，它在现实世界中是随机的。

这第三个特性是：确定不能再生成。即如果用完全相同的输入（至少尽可能的准确）运行两次序列生成器，将得到两个不同的随机序列。

另外，真随机序列不能压缩。密码安全伪随机序列实际上不能压缩。

具有这些特性的生成器的输出将对一次加密、密钥生成和其它所期望的密码特性已足够完善。

第3章 基本协议

3.1 密钥交换

普通的密码技术是用各自的密钥对两个人之间的某次对话进行加密。因为密钥仅用于通信会话，称为会话密钥。这个普通的会话密钥怎样交到对话者的手中倒是一件复杂的事。

对称密码术的密钥交换

(1)A 呼叫密钥分布中心(KDC)，请求一个同 B 通信的会话密钥。

(2)KDC 产生一个随机会话密钥。加密成两个复本：一个用 A 的密钥，另一个用 B 的密钥。KDC 也用 B 的密钥加密 A 的身份信息。KDC 将两者都送给 A。

(3)A 解密并得她的会话密钥。

(4)A 将 B 的会话密钥和她自己的身份信息送给 B。

(5)B 解密得到会话密钥和身份信息。他用身份信息判定 A 是谁。

(6)A 和 B 使用会话密钥安全地通信。

这个协议，取决于 KDC 的绝对安全。如果攻击者破坏了 KDC，整个网络将遭损坏，攻击者将与 KDC 共享每个用户的私钥；他只要搭上通信线收听加密信息就可知道所有来往的通信内容。

公钥密码术的密钥交换

(1)B 将他的公钥发给 A。

(2)A 生成一个随机会话密钥 K_1，用 B 的公钥加密后将它发给 B。

$E_B(K)$

(3)B 用他的私钥解密信息取得会话密钥。

(4)两者用相同的会话密钥进行加密通信。

用公钥数据库的公钥密码术的密钥交换

在一些实施中，A 和 B 的公钥可从数据库中拿到。这使密钥交换更为容易，即使 A 从没见过 B，也能发信息给他：

(1)A 从中心数据库取出 B 的公钥。

(2)A 生成一个随机会话密钥，用 B 的公钥加密，并发给 B。

(3)B 用他的私钥解密 A 的信息。

(4)两者用相同的会话密钥进行加密通信。

中间攻击

窃听者除试图攻击公钥算法或对密文作唯密文攻击外，不能再做什么。而攻击者能解密 A 和 B 之间的所有信息。他不但能听，也能修改信息、删除信息并产生新的信息。当同 A 谈话时，攻击者能冒充 B，当同 B 谈话时，他又冒充 A。这种攻击是如何工作的呢？

(1)当 A 把她的公钥给 B 时，攻击者截获这个密钥并将他自己的公钥发给 B。

(2)同样 B 把他的公钥发给 A。攻击者截获后也将他自己的公钥发给 A。

(3)当 A 发送以“B”的公钥加密的信息给 B 时，攻击者截获后。因为信息实际上用他自己的公钥加密，所以可用他的私钥解密，用 B 的公钥再加密，并发送给 B。

(4)当 B 发送以“A”的公钥加密的信息给 A 时，攻击者截获后同理可用他的私钥解密，再用 A 的公钥再加密，并发送给 A。

尽管 A 和 B 的公钥存在数据库中，这种攻击也将奏效。攻击者截获 A 对数据库的查询，用他自己公钥替代 A 的。对 B 也是一样。他秘密地打进数据库，用他的密钥替代 A 和 B 的。随后，他简单地等待 A 同 B 相互交谈，截获和修改信息。

因为 A 和 B 无法验证他们的相互谈话，中间攻击得以成功。假设攻击者处理快速而不产生任何明显的时延，A、B 两人不知道在他们之间正插入第三者在读他们所有想像中的安全通信。

防止这种攻击最容易的方式是让 A 和 B 确认他们使用相同的密钥。如果他们能打电话(或另一种允许他们有效地相互识别身份的方式)，则他们可相互告诉他们密钥的单向哈希值。遗憾的是这并非永远可行。

互锁协议

互锁协议由 Ron Rivest 和 Adi Shamir 发明，很好地破坏了中间攻击。其工作如下：

(1)A 将她的公钥发给 B；

(2)B 将他的公钥发给 A；

(3)A 用 B 的公钥加密她的信息。她把加密信息的一半发给 B；

(4)B 用 A 的公钥加密他的信息。他把加密信息的一半发给 A；

(5)A 把她加密信息的另一半发给 B；

(6)B 把 A 两个一半的信息的放在一起，用他的私钥解密。B 把他加密信息的另一半发给 A；

(7)A 把 B 两个一半的信息的放在一起，并用她的私钥解密。

要点是缺少信息的另一半，半个信息是无用的，不能解密。B 直到步骤(6)才能读 A 的信息；A 直到步骤(7)才能读 B 的信息。有许多方法实现这种协议：

- 如果加密算法是块算法，在每半个信息中将各块中的一半信息位发送出去。
- 信息的解密将取决于初始化矢量，矢量将同信息的另一半一起被发送。
- 前半个信息为加密信息的单向哈希函数，后半个信息为加密信息本身。

这就增加了攻击的麻烦，在步骤(1)和(2)攻击者用他自己的公钥替代 A 和 B 的公钥。但当他在步骤(3)截获 A 的一半信息时，他不能用他的私钥解密并用 B 的公钥再加密。他必须杜撰一个新的信息并发给 B 一半。其后，他截获步骤(5)和(6)的实际信息的第二个一半，延迟至使他不能改变他杜撰的新的信息。A 和 B 之间的对话将为全然不同。

有数字签名的密钥交换

在密钥交换中作数字签名可很好地防止了中间攻击的发生。KDC 签署 A 和 B 的公钥。签过名的密钥包括一个签有拥有权的证件。当 A 和 B 收到密钥时，他们各自验证 KDC 的签名。他们知道公钥属于谁。然后开始密钥交换。

这使攻击者碰到了严重的困难。因为他不知道他们任一个私钥，所以他不能模仿 A 或 B。他不能用他的公钥替代他们中的任一个，因为密钥被 KDC 签署。他只能听到加密信息来来去去或破坏通信线路阻止 A 和 B 的通信。

这个协议也使用一个 KDC，但 KDC 泄露的危险是非常小的。如果攻击者侵入 KDC，他取

得的是KDC的私钥。这个密钥仅能使他签一个新的密钥;不会让他解密任何会话密钥或读任何信息流。为了读信息流,攻击者必须能模仿网络上的一个用户,用他假冒的公钥加密的信息欺骗合法的用户。

攻击者能用KDC的私钥生成假冒的签名密钥来愚弄A和B。然后,同数据库中任一个实际签署的密钥交换,或截获用户的数据库询问并用他假冒的密钥回答。

但记住,截获和修改信息比被动地设置在对称网络中和读网络上通过的信息困难得多。主动搭线很难找到适当的隐蔽之处,非常容易被查出。在类似无线电一样的广播信道中,几乎不可能用另一个信息取代一个信息。

密钥和信息时戳

并非一定要A和B在信息交换之前完成密钥交换。在本协议中,A把信息M发给B,不用事先密钥交换:

(1)A生成随机会话密钥K,用K加密M;

$E_K(M)$

(2)A从数据库获得B的公钥;

(3)A用B的公钥加密K;

$E_B(K)$

(4)A把加密信息和加密会话密钥发给B。

$E_K(M),E_B(K)$

为加强安全,防止中间攻击,A签名传送。

(1)B使用他的私钥解密A的会话密钥K。

(2)B使用会话密钥解密A的信息。

这就是公钥密码术怎样用于通信系统的最普遍方式。它能把数字签名、时戳和任何其它安全的协议组合起来。

密钥和广播信息

A可以把加密信息发送给几个人。在这个例子中,A把加密信息发给B、C和D:

(1)A生成随机会话密钥K,用K加密M;

$E_K(M)$

(2)A从数据库获得B、C和D的公钥;

(3)A用B的公钥加密K、用C的公钥加密K,用D的公钥加密K;

$E_B(K),E_C(K),E_D(K)$

(4)A广播加密信息和把所有加密密钥广播给留心接收它的人;

(5)仅仅B、C和D能用他或她的私钥解密K;

(6)仅仅B、C和D能用K解密A的信息。

在存储转发网络中能实现这个协议。除他们实际的加密密钥外,中心服务器转发A的信息给B、C和D。服务器不必是安全的和可信的,因为他不能加密任何信息。

3.2 认　　证

当A登录到计算机时(例如,或自动语音、或电话银行系统),计算机怎样知道她是谁?计算机怎样知道她不是伪造身份的人?传统的方法是口令。A敲入她的口令,计算机确认它是否正确,人机均掌握正确的口令内容。A每一次登录,计算机都要求她输入口令。

一些密码学者认为，计算机实际上不需要知道口令；它仅须从无效的口令中区分有效的口令而已。用单向函数很容易来做到这一点。因此，计算机存储的是口令的单向函数值而不是口令本身。

(1)A 向计算机发送她的口令。

(2)计算机对口令执行单向函数运算。

(3)主机比较单向函数的结果和原有存储之值。

因为计算机不存储一张有效的用户口令表，侵入计算机并窃得用户口令的危险性可减轻。口令的单向函数表是无用的，因为从单向函数值不能反过来恢复口令。

查字典式攻击和“保护剂”

即使口令文件经单向函数加密，它还是易受攻击的。攻击者空闲时编制了一张 100,000 个普通口令的表格。他用单向函数运行 100,000 次并存储结果。如果每个口令 8 个字节，最终的文件正好为 800KB，存在一张软盘中。攻击者窃取了加密口令文件后，把加密口令文件与他的自制表格进行比较，看那个匹配。

这就是查字典式攻击，可意想不到地取得成功。“保护剂”是一种阻止查字典式攻击的方法。

在单向函数操作之前，“保护剂”是一个随机字符串，它同口令联系在一起由单向函数加密，“保护剂”的内容和单向函数加密后的结果都存储在数据库中。如果“保护剂”的可取内容数足够多，实际上就阻止了对普通口令的字典攻击。在初始化矢量中，这只是简单的目的。

需要有许多“保护剂”值。大部分 Unix 系统仅使用 12bit 的“保护剂”。即使那样，Daniel Klein 开发了一个口令猜测程序，大约一周内对给定的系统打开了 21%的口令。Daniel Klein 和 Phifp Karn 编制了一张大约 732,000 普通口令同“保护剂”的 4096 个可能值之一相联接的表格。估计对给定系统的 30%的口令能用它来攻破。

“保护剂”不是万能的，增加它的比特数也无济于事。事实上它仅仅用来防止口令文件的字典攻击，不能防止对单一口令的预定攻击。“保护剂”把在多台机器上用同一口令的人搞糊涂了，并不对选用欠妥的口令会有所改善。

用公钥密码的用户辩识

即使用“保护剂”，上面的协议还有严重的安全问题。当 A 输入口令进入系统时，访问她的数据库通道的人能读出口令。通过相关的传输通道访问她的计算机，诸如同行竞争者、国外人员或好奇的大学生借网络都可访问。在任何通道中的结点上窃听者能听到 A 的登录序列。此外，访问计算机系统内存储器的人能在系统加密之前看到口令。

公钥密码术能解决这个问题。主机保持每个用户的公钥的文件，所有用户保持他们自己的私钥。当登录时，协议以下面的方式进行处理：

(1)主机发给 A 一个随机字符串。

(2)A 用她的私钥加密字符串并与她的身份标志一同返回给主机。

(3)主机在数据库中寻找 A 的公钥，使用公钥解密信息。

(4)如果解密字符串与主机放在适当地方发给 A 的相匹配，主机允许 A 访问系统。

其他人无法取得 A 的私钥，所以难以模仿用户。更重要的是，A 从不需要把她的私钥通过传输线路发给主机。窃听交互过程，不能获得任何信息，无法得到私钥和模仿 A。

私钥长而难于记忆，可通过用户的硬件或通信软件自动地处理。但需要一个 A 信任的智能终端，这将带来主机与通信信道之外的安全要求。

一般地说，用户去加密另一方而不是主机发送的随机字符串是愚蠢的；这会招至攻击。

安全身份证明协议的形式如下：

(1)A 依据一些随机数和她的私钥完成计算，并把结果发给主机。

(2)主机发给 A 一个不同的随机数。

(3)A 依据随机数(她产生的和从主机接收的)和她私钥进行计算并把结果发送主机。

(4)主机依据从 A 接收的各种数据和她的公钥进行计算验证她知道自己的私有密钥。

(5)如果是这样，她的身份得到验证。

如果 A 就象主机不相信 A 一样不信任主机，那未 A 将需要主机提供他的身份证明。

步骤(1)看上去像是不必而又使人不解，其实这是协议防止攻击的要求。

使用互锁协议相互证实

A 和 B 是想要相互证实的两个用户。各有一个口令：A 的为 P_A 和 B 的为 P_B。如下协议是不行的：

(1)A 和 B 交换公开的密钥。

(2)A 用 B 的公钥加密 P_A 并发给他。

(3)B 用 A 的公钥加密 P_B 并发给她。

(4)A 解密 P_B 并验证为正确的。

(5)B 解密 P_A 并验证为正确的。

因为攻击者可产生一个成功的中间攻击：

(1)A 和 B 交换公钥，攻击者截获两个信息。他用他的公钥替代 B 的并发给 A，用他的公钥替代 A 的并发给 B。

(2)A 用“B”的公钥加密 P_A 并发给 B，攻击者截获信息，用他的私钥解密 P_A，用 B 的公钥再加密并发给 B。

(3)B 用“A”的公钥加密 P_B 并发给 A，攻击者截获信息，用他的私钥解密 P_B，用 A 的公钥再加密并发给 A。

(4)A 解密 P_B 并验证为正确的。

(5)B 解密 P_A 并验证为正确的。

从 A 和 B 看到的没有什么不同。然而，攻击者知道了 P_A 和 P_B。Davies 和 Price 提出了互锁协议来克服这类攻击：

(1)A 和 B 交换公开的密钥。

(2)A 用 B 的公钥加密 P_A 并发送它的一半给他。

(3)B 用 A 的公钥加密 P_B 并发送它的一半给她。

(4)A 把加密 P_A 的另一半发给 B。

(5)B 把两个一半合并在一起，解密 P_A 并验证为正确的。

(6)B 加密 P_B 的另一半发给 A。

(7)A 把两个一半合并在一起，解密 P_B 并验证为正确的。

也有人讨论了攻击这个协议的方法。如果 A 为用户而 B 为主机，攻击者假装为 B，完成与 A 相关的协议的步骤(1)到步骤(5)并结束连接。攻击者成功的真正技巧在于用模拟在线噪声和网络失效的方法去撇开实际的主机，而让他得到 A 的口令。他能与 B 连接并完成协议。

在用户的口令比主机的口令更易暴露的前提下，把协议修改成 B 事先把他的口令交给 A，可减少更复杂的攻击。

SKID

SKID2 和 SKID3 是开发 RIPE 方案的私钥身份协议。他们使用一个加密密钥单向哈希函数(MAC)来提供安全,并假设 A 和 B 共享一个私钥 K。

SKID2 允许 B 向 A 证明他的身份。协议为:

(1)A 选择一个随机数 R_A(RIPE 文件规定为一个 64 比特数字),发给 B。

(2)B 选择一个随机数 R_B(RIPE 文件规定为一个 64 比特数字),发给 A:

$R_B, H_K(R_A, R_B, B)$

H_K 为 MAC。

(3)A 计算 $H_K(R_A, R_B, B)$,同从 B 处接收的进行比较。如果结果是相同的,A 知道她正与 B 通信。

SKID3 提供了 A 和 B 之间的相互证实。步骤(1)至(3)同 SKID 相同,然后协议进入。

(4)A 发给 B:

$H_K(K_B, A)$

(5)B 计算 $H_K(K_B, A)$,同从 A 那里接收的进行比较。如果结果是相同的,B 知道他正与 A 通信。

3.3 证实与密钥交换

这些协议解决一个计算机的普通问题:A 和 B 位于网络的不同端,他们想安全地通话。A 和 B 怎样交换密钥,同时确信是他们在互相谈话而不是同攻击者谈话?

阔口青蛙

阔口青蛙协议可能是最简单的对称密钥管理协议,协议使用一个可信的服务器。B 和 A 与仲裁人 T 共享私钥。假设信道为安全的,会话密钥仅用于密钥分布,不加密用户之间的任何实际信息。会话密钥从 A 传送到 B 仅用两个信息:

(1)A 联接时戳 T_A、B 的名字和随机会话密钥 K 成一体,用她与仲裁人共享的密钥加密全部信息。她把信息同她的身份 A 一起发给仲裁人。

$A, E_A(T_A, B, K)$

(2)仲裁人联接时戳 T_B、A 的名字和随机会话密钥 K 成一体,用他与 B 共享的密钥加密全部信息。仲裁人把它发给 B:

$E_B(T_B, A, K)$

协议中最重要的假设是 A 足够的能力产生好的会话密钥。

Yahalom 协议

它由 Yahalom 提出。与前面的协议一样,A 和 B 同仲裁人 T 共享一个私钥。

(1)A 将她的名字与随机数 R_A 联结在一起并发给 B。

A, R_A

(2)B 又将 A 的名字、她的随机数和另一个随机数 R_B 联结在一起,用他与仲裁人共享的密钥加密它。B 把这个信息同他的名字一起发给仲裁人。

$B, E_B(A, R_A, R_B)$

(3)仲裁人产生两个信息。第一部分包含 B 的名字、A 和 B 的随机会话密钥 K、A 的随机数、B 的随机数,用与 A 共享的密钥加密所有信息。第二部分包含 A 的名字和随机会话密钥,用他与 B 共享的密钥加密。他把两个信息发给 A:

$E_A(B,K,R_A,R_B),E_B(A,K)$

(4)A 解密用她的密钥加密的信息，提取 K，确认 R_A 与步骤(1)有相同的值。A 发给 B 两个信息。第一个信息为从仲裁人处接收的、用 B 的密钥加密的信息。第二个为用会话密钥加密的 R_B。

$E_B(A,K),E_K(B)$

(5)B 解密用他的密钥加密的信息，提取 K，确认 R_B 与步骤(2)所做的有相同的值。

最后，A 和 B 信任他们正相互谈话并没有第三方。其特色是 B 是第一个与仲裁人接触，而仲裁人仅把信息发给 A。

Needham 和 Schroeder 协议

Needham 和 Schroeder 提出了另一个协议，它也使用对称密码术和仲裁人 T。

(1)A 把含有她的名字、B 的名字和一些随机值 R_A 的信息发给仲裁人。

(A,B,R_A)

(2)仲裁人产生一个随机会话密钥 K。他用与 B 共享的私钥加密包含 K 和 A 名字的信息。他用与 A 共享的私钥加密 A 的随机值、B 的名字、密钥和已被加密的信息。最后他把加密信息发给 A。

$E_A(R_A,B,K,E_B(K,A))$

(3)A 解密信息并提取 K。她确认 R_A 与在步骤(1)她发给仲裁人值相同。

$E_B(K,A)$

(4)B 解密信息并提取 K。他产生另一个随机值 R_B。他用 K 加密信息并发给 A。

$E_K(R_B)$

(5)A 用 K 解密信息。产生 R_B-1 并用 K 加密。然后把信息送回给 B。

$E_K(R_B-1)$

(6)B 用 K 解密信息并证实为(R_B-1)

围绕 R_A、R_B 和 R_B-1，令人高兴的是保证没有应答攻击。在步骤(3)R_A 的出现使 A 确信仲裁人的信息是合法的，并非是协议前次执行时的回答。同样当 A 成功地解密 R_B 同时在第五步将 R_B-1 送给 B 后，B 也确信 A 的信息并非是协议前次执行的回答。

这个协议安全上最大的问题是原有的会话密钥易受攻击。如果攻击者访问到原有的 K，他能产生成功的攻击。他所要做的是记录步骤(3)A 到 B 的信息。他一旦得到 K，他就能冒充为 A：

(1)攻击者发给 B 如下的信息：

$E_B(K,A)$

(2)B 提取 K，生成 R_B，并发给“A”：

$E_K(R_B)$

(3)攻击者截获信息，用 K 解密并发给 B：

$E_K(R_B-1)$

(4)B 验证“A”的信息为 R_B-1。

现在，攻击者使 B 相信他是 A。

一个更强有力的协议是使用时戳来防止这种攻击。这个协议需要一个安全的和准确的系统时钟，这不是一件小事。这个修改的协议是 Kerberos 证实协议的基础。

如果 K_A 一旦泄露，结果更为严重。攻击者能使用它得到同 B 谈话(或他也想同任何人谈

话)的会话密钥。更糟的是,即使 A 改变她的密钥后,攻击者能继续这样做。

Needham 和 Schroeder 试图在他们协议的修改版中改正这些问题。他们的新协议基本上与 Otway-Rees 协议相同,并在同时同刊发表。

Otway-Rees 协议

Otway-Rees 协议也使用对称密码术。像通常一样,仲裁人 T 同网络上的任何人共享一个独立的密钥。

(1)A 产生一个包含索引号 I、她的名字、B 的名字和随机数 R_A 的信息,用与仲裁人共享的密钥加密所有信息。她把加密信息同索引号、她和对方的名字一起发给 B:

$$I,A,B,E_A(R_A,I,A,B)$$

(2)B 产生一个包含索引号 I、他的名字、A 的名字和随机数 R_B 的信息,用与仲裁人共享的密钥加密所有信息。他把加密信息同 A 加密信息、索引号、双方名字一起,发给仲裁人:

$$I,A,B,E_A(R_B,I,A,B),E_B(R_B,I,A,B)$$

(3)仲裁人产生一个随机会话密钥 K。然后他产生两个信息。一个为 A 的随机数和会话密钥,用与 A 共享的密钥加密。另一个是 B 的随机数和会话密钥,用与 B 共享的密钥加密。他把这两个信息与索引号一起,发送给 B:

$$I,E_A(R_A,K),E_B(R_B,K)$$

(4)B 把与索引号一道用她的密钥加密的信息发给 A:

$$I,E_A(R_A,K)$$

假设所有的随机数匹配,索引号沿袭不变,A 和 B 互信各自的身份,他们具有通信的私钥。

Kerberos 协议

Kerberos 协议已经实现,基本的 Kerberos 版本与协议如下:A 和 B 与仲裁人 T 共享密钥。A 想产生一个同 B 通话会话密钥。

(1)A 把他的身份、B 的身份的信息发给仲裁人。

$$A,B$$

(2)仲裁人产生一个具有时戳 T、生命 L、随机会话密钥 K 和 A 身份的信息。他用与 B 共享的密钥加密这个信息。他取出时戳、生命、会话密钥和 B 身份,用与 A 共享的密钥加密这个信息。

$$E_A(T,L,K,B),E_B(T,L,K,A)$$

(3)A 产生一个具有她的身份和时戳的信息,用 K 加密它,把它发给 B。A 也把来自仲裁人按 B 的密钥加密的信息发给 B。

$$E_K(A,T),E_B(T,L,K,A)$$

(4)B 发送一个用 K 加密时戳加 1 的信息,并发给 A。

$$E_K(T+1)$$

为协议可行,要假设每一个人的时钟与仲裁人的时钟同步。实际上,协议用了同步时钟。

SPX 协议

SPX 协议由 DEC 公司开发,也提供相互证实和密钥交换。与其他协议不同,SPX 使用公钥和对称密码术。A 和 B 每一个人有一个私钥。仲裁人 T 在他们公钥的拷贝上签了字。

(1)A 把包含 B 的身份信息发给仲裁人。

(2)仲裁人把 B 的公钥 K_B 发给 A,它有仲裁人的私钥,T 的签名。

$$S_T(K_B)$$

(3)A 验证仲裁人的签名，确认她收到的密钥实际上是 B 的公钥。她产生一个随机的秘密的密钥 K 和一对随机的公钥/私钥 K_P。她用 K 加密时间 T_A。她用她的私钥 K_A 对密码生命 L、她的身份 A 和 K_P 签名。最后，她用 B 的公钥加密 K，用 K_P 对它签名。她把所有信息发给 B。

$E_K(T), S_{KA}(L, A, K_P), S_{KP}(K_B(K))$

(4)B 把包含 A 身份的信息发给仲裁人（这可能是另一仲裁人）。

A

(5)仲裁人把 A 的公钥发给 B，并用仲裁人的私钥签名。

$S_T(K_A)$

(6)B 验证仲裁人的签名，确认他收到的密钥实际上为 A 的公钥。他验证 A 的签名并恢复 K_P。他验证签名，使用他的私钥来恢复 K。然后，他解密 T_A 使其确信这是目前的信息。

(7)如需要相互证实，B 用 K 加密一个新的时戳 T_B，并发给 A。

$E_K(T_B)$

(8)A 用 K 解密 T_B，使其确信这是目前的信息。

DEC 有一个正在实施的 SPX 协议。

3.4 多密钥、公钥密码学

在公钥密码学中有两个密钥。用一个密钥加密的信息能被另一个密钥解密。通常，一个密钥是秘密的，而另一个为公开的。于是，假设 A 有一个密钥，B 有另一个密钥。A 加密的信息仅能由 B 来解密，B 加密的信息仅能由 A 来解密。

Colin Boyd 给出了上述概念。设想一个有三个密钥的公钥密码方法的变例。K_A、K_B 和 K_C。A 有第一个，B 有第二个，C 有第三个。另外，D 有 K_A 和 K_B。E 有 K_B 和 K_C。F 有 K_C 和 K_A。任意个密钥能被用于加密。其余的密钥则用来解密。

A 用 K_A 加密信息，因此 E 用 K_B 和 K_C 解密，就像 B 和 C 结成一体一样。B 加密信息，结果是 F 能读，C 加密信息结果是 D 能读。D 用 K_A 加密信息，结果是 E 能读，用 K_B 加密结果是 F 能读，或用 K_A 和 K_B 加密，结果是 C 能读。类似地，E 加密信息，结果是 A、D 或 F 中任一个人都能读。其它配对不能进行通信。这些关系小结如表 3.1 所示。

这种关系同样能扩展至 n 个密钥。如果密钥的某一子集用于加密信息，其余的密钥则用来解密信息。

表 3.1 三密钥信息加密

用于加密的密钥	必须用于解密的密钥
K_A	K_B 和 K_C
K_B	K_A 和 K_C
K_C	K_A 和 K_B
K_A 和 K_B	K_C
K_A 和 K_C	K_B
K_B 和 K_C	K_A

广播信息

设想有三个操作者：A、B 和 C。你想发送信息给他们中的一个子集，但无法预先知道哪一个子集。你需单独地为每人加密信息或对每一可能的组合给出密钥。前者通信流量大；后者需要许多密钥。

多密钥密码术就比较简单了。当你将 K_A 和 K_B 给予 A，K_B 和 K_C 给予 B，K_C 和 K_A 给予 C 之后。就能同你想要谈的任何子集谈话。如果你想发送一个信息仅使 A 读它，用 K_C 加密。当 A 接收信息时，她用 K_C 然后用 K_B 进行解密。如果你想发送信息仅使 B 能读，用 K_A 加密；仅使 C 能读，用 K_B 加密。如果你想发送信息能让 A 和 B 能读，用 K_A 和 K_C 加密，以次类推。

对三人而言这似乎只是平平，但 100 个操作者就能体会到这个方案的简易程度。如果你想把信息发给操作者的任一子集，你有三种选择：同每个操作者（总共 100 密钥）共享一个密钥并发送单独的信息；分送 $2^{100}-1$ 个密钥给每一个可能子集的用户。或使用如下方案：以一个加密的信息和 100 个不同的密钥来完成。

3.5 密钥的拆分

假若你发明了一个配方，生产中保密的方法是不让生产者知道它包含的所有成分，每个生产者只知道一、二样，即使泄露了也无关大局。同样的道理用于密码术称为密码拆分。将所取信息分成多片。每一片本身无任何意思，但把他们放在一起，意义就明白了。

最简单的共享问题是在两个人之间分解信息。仲裁人在 A 和 B 之间分解信息的协议为：

(1)产生一个随机比特串 R，与信息 M 长度相同。

(2)把 M 与 R 异或产生 P

$$M \oplus R = P$$

(3)把 P 交给 A，并把 R 交给 B。

再构造信息，A 和 B 仅有一步可做；

(4)A 和 B 把他们的小片异或在一起构成信息：

$$P \oplus R = M$$

这种技术如能实现，是绝对安全的。每一单片毫无价值，仅仅结合在一起，A 和 B 能再构成信息。

基本上，仲裁人用一次一密加密器加密信息，把密文交给一个人而把加密器交给另一个人。他们有非常好的安全性。无需大量的计算就能决定信息来自哪一片。

把这个方案扩展到多人也是容易的。把信息分解成多份，异或上更多的随机比特串形成一个混合体。在下例中，仲裁人把信息划分为四段：

(1)产生三个随机比特串 R、S 和 T，与信息 M 长度相同。

(2)把 M 与三个比特串异或产生 P：

$$M \oplus R \oplus S \oplus T = P$$

(3)把 P 交给 A，R 交给 B，S 交给 C，T 交给 D。

这里是 A、B 和 C 再构造的信息。

(4)A、B 和 C 结合在一起并计算：

$$P \oplus R \oplus S \oplus T = M$$

这是一个裁决协议。仲裁人具有绝对能力做他想做的任何事。他能将无用的数据宣称是密码的有效片段；直至试图再构成密码时没人会明白。例如他将数据库送给 A、B、C 和 D，随后，告诉每一个人仅需要 A、C 和 D 的部分就能构成密码，然后他毁掉了 B 的那一片。这都是由仲裁人秘密划分决定的不会有任何问题。

然而，如果丢掉了一片数据，怎样再构成信息就产生了问题，下一节将讨论如何解决。

3.6 秘密共享

复杂的秘密共享可用门限方式，将任何信息(如配方、密钥、口令等)分成 n 片影条，并使得取用任意 m 片影条，就可构成原始信息，它又称为(m、n)门限方案。它由 Adi Shamir 和 G. R. Blaklay 分别提出。

如(3,4)门限方案，仲裁人把他的秘密配方分给 A、B、C 和 D，使得任何三个把他们的影条放在一起，能重新得到配方。如果 C 在渡假，A、B 和 D 能组成配方。如果 B 开车旅行去了，A、C 和 D 能完成。然而，如果 B 开车旅行而 C 在渡假，A 和 D 就不能再构造信息。

门限方案还有更灵活的地方。任何共享的情况能用它来构造。

共享秘密中的欺骗

门限共享方案中会遭受到各种欺骗。这里仅有几例：

例 1：A、B、和 C 在某深深的掩体内。一天，他们得到来自总统的密令："发射那些导弹。我们依据国家网络中枢的最后证据继续发射。"A、B、和 C 展现他们的影条，但 C 进行欺骗，随意键入。因为他是一个和平主义者，不想发射导弹。由于 C 键入不正确的影条，恢复成错误的密码。导弹留在发射井中。没人知道原因。即使他们在一起工作，A 和 B 不能证明 C 的影条是不合法的。

例 2：A 和 B 与攻击者正位于掩体中。攻击者把自己扮成他们中的一员，事先他们并不认识。同样的信息来自总统，每一个人展现他们的影条。攻击者用伪造的信息，知道了其他二人的影条。

例 3：A、B、和 C 与伪装后的攻击者，正位于掩体中。同样的信息来自总统，每人展现他们的影条。攻击者仅在听到其它三个人的影条之后，再展现他的影条。因为其它三个人的影条已能恢复原始信息，他可用来产生一个有效的影条然后展现它。攻击者不但知道了密码，而且可继续伪装不被识破。

3.7 数据库的密码保护

专业机构的会员联络数据库颇有商品价值，数据库提供给所有成员使用。他们能相互通信、交换观点、举行茶话会。但另一方面，如果你把会员数据库的拷贝分送给每个人，难免会落入保险推销员和专送无用邮件的干扰者手中而带来麻烦。

密码术能改善这种情况。即以个人地址易访，全体成员的邮址难得的方式加密数据库。

这个方案选用了一个单向哈希函数和一个对称加密算法。数据库的每一个记录包含两个域。索引域是成员的姓，按单向哈希函数操作。数据域是成员的全名和地址，用姓作为密钥将其加密。

给定姓氏搜索成员是容易的。首先，对姓作哈希运算，寻找数据库中索引域的哈希值。如果匹配，所访之人即为此姓。然后，对每一匹配者，以姓为密钥解密全名和地址。

这种保护不是完美的。实际上，耐心的销售者通过试验每一可能的姓氏可重构会员数据库。用电话数据库做参考，计算机运行上周时间，虽然复杂，但也能做到。

3.8 时戳服务

许多情况人们需要验证文件产生于某时。考虑版权或专利的争端，能提供争端内容最早副本的一方将能胜诉。用纸面文件有许多方法，公证人签署文件；律师安全地保留拷贝。如果争

端出现，公证人或律师提供证据，文件是在某日产生的。

在数字世界中，这将非常复杂。无法检查数字文件的签名有否篡改。无需任何智慧就可无尽止地拷贝和修改文件。在数字文件中改变时戳是小事一桩。没有人去检查数字文件的日期并相信“是的，这个文件在 1942 年 12 月 10 日之前生成的。”

Stuart Haber 和 W。Scott Stornetta 思考这个问题。他们发明了有如下三个特性的数字时戳协议：

- 日期本身必须为时戳，无论用何种物理介质存储，数据必须带有时戳。
- 如果文件外观不变化，而企图改变文件乃至仅一位也是不可能的。
- 用与实际文件不同的日期和时间去加时戳是绝不可能的。

仲裁方案

这个协议使用仲裁人，他具有可信任的时戳服务。当然，A 事先要在文件上打有时戳。

(1)A 把文件的拷贝传给仲裁人。

(2)仲裁人记录他收到文件的日期和时间，留有需安全保护的文件拷贝。

如果任何人对 A 声明的文件时日有所查询，她必须请求仲裁人。让仲裁人产生一个文件拷贝，验证他掌握的时戳。

这个协议操作，有明显的问题。首先，没有保密，A 必须把文件的拷贝给仲裁人。在通信信道上收听的人能读取。她可加密，但文件仍然位于仲裁人的数据库中。数据库的安全性如何？

第二，数据库本身将是大容量的，因为存储需求是如此的大。发送大量文件给仲裁人，需求的带宽是非常大的。

第三，可能产生潜在的错误。在仲裁人的中心计算机中，传输的错误或某处磁带的损坏将完全地使 A 时戳申请变为无效。

第四，可能有一些人不像仲裁人那样诚实地使用时戳服务。A 可以使用 B 的时戳。无法阻止 A 和 B 串通一起用他们所要的时间打上时戳。

改进的仲裁方案

单向哈希函数和数字签名能方便地用来解决大部分问题：

(1)A 产生文件的单向哈希值；

(2)A 把哈希值传给仲裁人；

(3)仲裁人把他收到的时间和日期作哈希运算再附加到哈希值上，然后在结果上完成数字签名；

(4)仲裁人把签名后的哈希结果和时戳返回 A。

这解决了除上述之外的所有问题，A 不再担心展现她文件的内容了；哈希是充分有效的。仲裁人不再存储文件的拷贝(甚至其哈希值)，解决了大量的存储需求和安全问题(记住，单向哈希函数没有密钥)。A 能立即检查在步骤(4)收到的签名时戳，使得她能立即捕捉到任何传输错误。所剩的问题是 A 和仲裁人仍可串通来产生他们想要的任何时戳。

连接协议

解决这个问题的一种方法是把 A 的时戳与以前收到的时戳连接起来。因为仲裁人收到的不同时戳请求的顺序不能预先知道，所以 A 的时戳必须发生在前一个之后。因为后来的请求与 A 的时戳连接，她的时戳必须在此之前发生。A 的请求时间夹在两者之中。

如果 A 想时戳的哈希值为 H_n，以前的时戳为 T_{n-1}，则协议为：

(1)A 把 H_n 和 A 发给仲裁人。

(2)仲裁人返回给 A：

$T_n=S_K(n, A, H_n, T_n; ID_{n-1}, H_{n-1}, T_{n-1}, H(ID_{n-1}, T_{n-1}, H_{n-1}))$

(3)A 的名字表示她是请求的原主。参数 n 表示请求的序列;仲裁人发出的第 n 个时戳。参数 T_n 是时间。另外的信息是身份、原始哈希和仲裁人打下的前文件哈希运算后的时戳。

(4)仲裁人将下一个文件加盖时戳后,他把文件原主的身份(ID_{n+1})发给 A。

如果某些人对 A 的时戳质疑,她只需同前后文件的原主身份 ID_{n-1}和 ID_{n+1}对照,如果他们的文件产生问题,他们能同 ID_{n-2}和 ID_{n-2}取得联系,依次类推。每一个人能显示他们的文件的时戳,无论是先来后去,还是先去后来。

这个协议使 A 和仲裁人串通并产生一个与实际不同的时间时戳文件变得非常困难。因为无法预先知道文件请求次序,所以仲裁人不能预发时间给一文件。即使伪造了也不行,因为时戳必须嵌在立即产生的文件的时戳中,文件已经被产生后,他也不能在文件上倒退时间。攻破这个方案唯一可能的方式是在 A 文件之前和之后,发明一个假想的文件链,但它足以耗尽任何人的精力去解决时戳。

分布协议

人逝则时戳不复存在。在给出时戳和 A 得不到 ID_{n-1}的时戳拷贝之间,许多事情都会发生。如果将前十个人的时戳嵌入到 A 的时戳之中,然后再把随后十个人的 ID 送给 A,这个问题可缓解。A 有较多的机会去找出仍有其时戳的人们。

(1)以 H_n 作为输入,A 使用一个密码安全的伪随机数生成器生成一个随机数值串:

$V_1, V_2, \cdots\cdots V_k$

(2)A 将每一个值作为另一个人的 ID,他把 H_n 发给这些人中的每一位。

(3)每个人将附属日期和时间附加到哈希值上,签署结果,返回到 A。

(4)A 收集和存储所有带时戳的签名。

步骤(1)密码安全的伪随机数生成器防止 A 选用可舞弊的 ID 作为验证符。即使她在她的文件中做很小的变化,试图构成一批可舞弊的 ID,她要摆脱该协议是几乎不可能的。哈希函数随机化了 ID,A 不能强行置换们。

这个协议能很好地工作,因为 A 伪造时戳的唯一方式是要说服所有的人(K 个)都与她合作。因为她在步骤(1)随机地选择他们,不愿入伙的可能性是非常高的。社会越腐败,数字 k 要取得越大。

此外,还需有一些机构使人们不能立即返回时戳,要求 K 中的某些子集具有有效时戳,详细情况取决于具体实施。

进一步研究

时戳协议的改进。有使用二叉树增加时戳量,甚至进一步推出产生假想的时戳链的可能性。他们也建议在公开的地方发布每天时戳的哈希值,其功能类似于在分布数据库中把哈希值发给任选者。事实上,自 1992 年以后,BELLCORE 在每周六的“纽约时报”上做这件事。

第4章 中级协议

4.1 难以觉察的信道

设想A和B已被捕入狱，分处男女牢房，他们仅有的通信方式是通过看守人传递条子。看守人不允许他们加密，因为他要知道相互通信的内容。

同时看守人也期望能欺骗A和B，使他们把条子上的假信息认为是来自另一方的真信息。A和B情愿冒着被欺骗的风险，否则他们几乎不能通信。于是他们调整计划，建立一个难以觉察的信道，他们之间的秘密通信信道。在看守人眼中，仅是完全乏味的信息交换，不含有秘密。然而，正是这种貌似平常的内容传递，A和B完成了秘密通信，看守人即使监视着整个通信过程，他还是被愚弄。

在句子中，一个易行的难以觉察的信道可能为数字。句子包含偶数个词则表示为“1”，而奇数个词相当于“0”。于是，当读了似乎乏味的三句语句后，籍它已经发送了“101”的指令。这类算法没有密钥，安全性取决于算法的安全。

Peter Wayner 的模拟函数使信息模糊。这些函数通过对信息的修改隐蔽了原貌，使它原文看起来类似莎士比亚的剧本或某一报刊的文章。它没有相应的密钥，这是一个很局限的算法。

Gustavus Simmons 在通常的数字签名算法中发明了难以觉察的信道的概念。因为难以觉察的信道是隐藏在看似普通的数字签名中，这是一种迷惑人的方法。事实上，难以觉察信道的数字签名与通常的数字签名没有任何区别。

协议一般为如下形式：

(1)A 随机产生一个乏味的信息。

(2)A 用与B共享的私钥签署乏味的信息，方式为把她难以觉察的信息隐藏在签名中。

(3)A 通过看守发送这个签名信息给B。

(4)看守读乏味的信息并检查签名，没发现毛病，他把签名信息传给B。

(5)B 检查乏味信息的签名，确认信息来自A。

(6)B 不管乏味的信息，使用与A共享的私钥，提取难以觉察的信息。

什么是欺骗？看守不信任任何人，也没有犯人信任他。他虽常要妨碍通信，但无法加入伪造的信息。因为他不能产生任何有效的签名，B在步骤(5)将检查看守有否做假。由于看守不知共享的密钥，不能读难以觉察的信息。更重要的是他无法知道难以觉察的信息在哪里。使用数字签名算法签名的信息看上去与用嵌在签名中的难以觉察的信息的签名息没有区别。

A和B之间进行欺骗更是一个问题。在一些难以觉察的信道的实施中，B需得到的秘密信息与A签署的秘密是相同的。此时，B能冒充A并假造来自A的信息，而实际上A并没有做。所以如果A要把难以觉察的信息发给B，她必须信任B而不乱用A的私钥。

其它的难以觉察的信道的实施没有这个问题。A和B共享的私钥允许A把难以觉察的信息发送给B，但与A的私钥不同，不允许B签署信息。A不必确定B不乱用她的私钥。

难以觉察的信道的应用

难以觉察的信道的最明显的应用是间谍网络。如果每个人收发经签名的信息，在这类签名文件中夹带间谍发送隐蔽信息是不会被注意的。当然，敌方的间谍也能同样作为。

使用难以觉察的信道，A 在胁迫下能安全地签署文件。当签文件时，他嵌入难以觉察的信息，说："我正被强制做某事"。其它的应用更是层出不穷。公司能签署文件并嵌入难以觉察的信息，对文件流通的全过程进行跟踪。政府能对数字货币加上标志。模糊的签名程序能告知秘密的密钥。各类可能的应用领域难以一一列出。

不夹带秘密的签名

A 和 B 相互发送签名信息，谈判合同的用词。他们使用数字签名协议。然而，这个合同的谈判已成为 A 和 B 间谍活动的伪装。当他们使用数字签名算法时，并不关心他们正签名的信息。他们使用难以觉察的信道以签名的方式相互发送秘密信息。然而，反间谍机关不知道合同谈判和使用签名信息仅是一种掩盖手段。

难以觉察的信道的使用让人们产生了不夹带秘密的签名方案。这些数字签名方案，不能有难以觉察的信道。

4.2 不可否认的数字签名

某软件公司发布了一个新软件。为确保他们的软件无病毒，上面包含数字签名。然而，仅为软件的合法买方，不是非法翻版，能验证签名。同时，如果发现新软件的拷贝含有病毒，该软件公司无法拒绝其有效的签名而承担责任。

通常的数字签名实际上是能复制的。有时，这个特性是有用的，如传播公开的布告。但在某些时候却是问题。试想，在商务或个人信件中的数字签名，如果复制件满天飞，每一份都当是真的，这也使人窘迫或导致黑信泛滥。最好的解决方案是能判明数字签名的有效性，但未经签名者同意，收信者不能向第三方显示。

由 David Chaum 发明的不可否认的签名适合以上任务。像通常的数字签名，不可否认的签名取决于签名文件和签名者的私钥。但不同于通常的数字签名，未经签名者同意，不能验证不可否认的签名。

协议的基本思想简述如下：

(1)A 为 B 提供一个签名；

(2)B 产生一个随机数并发给 A；

(3)A 使用随机数和她的私钥计算，并把结果发给 B。仅在签名有效时，A 可进行计算。

(4)B 加以确认。

B 不能使另一人信服 A 的签名是有效的，因为他不知道 B 的数字为随机的。B 从纸上倒行协议是易行的，无需 A 的任何帮助，将结果显示给另一人，但不能使他相信 A 的签名有效，除非同 A 本人按协议进行工作。

这个方案并不完善。Yvo Desmed 和 Moti Yung 证明在一些应用中，B 可使另一人信服 A 的签名有效，这是一种例如国际象棋大师问题的攻击。

4.3 失败—终止式数字签名

假如攻击者是个强劲的对手，他有巨大的计算机网络，许许多多巨型机，其计算能力高于 A 几个数量级，在这样的装备下，计算机日夜工作，A 的私钥被攻破了，攻击者能模仿 A，在以

后的文件上伪装他的签名。

由 B. Pfitzmann 等引出的失败—终止式数字签名防止了这类欺骗事件，当攻击者伪造了签名后，A 能给出证据说明它们是伪造的，而对 A 自己所做的签名，法院能证明这不是伪造的。

失败—终止签名的基本想法是：

每一个可能的公钥，可有许多同它一起工作的私钥。每一个私钥可产生许多不同的签名。然而 A 仅有一个密钥而且只计算一个签名，A 不知道私钥中的任何一个其他的密钥。

攻击者试图破译 A 的私钥。通过收集签名信息，使用阵列机，恢复了私钥。但由于可能私钥之多，很可能她发现了一个不同的密钥。攻击者攻击成功的可能性是如此之小以致忽略不计。

当攻击者使用产生的私钥伪造签名文件时，将是一个不同的签名文件而不是 A 她自己签名的文件。对法庭，A 能对同一的信息和公钥产生两个不同的签名，从而证明有伪。换句话说，如果 A 不能产生两个不同的签名，即没有伪造，她的签名必定为 A 的。

这种签名方案防止了攻击者用绝对的计算能力来破译 A 的签名方案；当然它无力防止盗窃或签名者伪造报失之类的攻击。

4.4 组签名

David Chaum 引入这个问题：

公司有几台计算机，每台都连在局域网上。公司的每一部门有它专用打印机（也连在网络上），属于某一部门的人仅仅允许使用他们部门相应的打印机。因此，在打印前必须使打印机确定用户正在那个部门工作。同时，公司为着隐私，并不显示用户的名称。然而，发现某人一天用得太多时，主任必须能确定是谁并送一张罚款单给他。

这个问题的解决方法称为组签名。组签名有以下特性：

- 仅仅组员能签署信息。
- 签名的接收者能验证有效的签名来自哪个组，但不能决定是组里的某人签名。
- 有争端时，签名能“打开”，展现出签名者的身份。

有可信仲裁者的组签名

这个协议使用一个可信的仲裁者：

(1)信托人产生大量公钥/私钥对，给每一组员一张不同的私钥表格。表中没一个密钥是相同的。

(2)信托人公布全组所有公钥的主表，它按随机顺序列出。信托人保留一份密钥属主的秘密记录。

(3)当组员想签署文件时，他们随机地从他们的个人表格中选出一个密钥。

(4)当他们想验证签名是否属于本组之中时，检查公开数据库相应的公钥并验证签名。

(5)有争端时，信托人知道哪个公钥对应哪一个组员。

该协议要求信托人是值得信任的。她知道每个人的私钥，能伪造签名。所有成员必须有一张足够长的密钥表以防止破译所用密钥的企图。

Chaum 的论文列出了许多其它协议，一些信托人不能伪造签名，而另一些甚至不需要信托人。

4.5 计算加密数据

对一些特殊的 X 值，A 想知道函数 F(X)的结果。不幸的是她的计算机坏了。B 乐意为她计算 F(X)，但 A 不愿让 B 知道她的 X。A 怎样让 B 为她计算 F(X)，而不把 X 告诉他？

这是数据加密计算的一般形式，也称为来自神殿的隐蔽信息。对某些函数，有方法做到上述要求。

4.6 位 约 定

现实生活中常有这样的问题，例如：A 给出一个预言，但不想在某时之前显示给 B。另一方面，B 也想确定 A 做出预言后，并没有修改。为此可用位约定。

传统的密码术的位约定协议

(1)B 产生随机位串 R，并发给 A。

(2)A 生成一个信息，它包含约定的位 b(实际上是几位)和 B 的随机串。她用随机密钥 K 加密信息，把结果返回 B。

$E_k(R,b)$

这是协议的约定部分。B 不能解密信息，他不知道这些位的内容是什么。

A 展示位内容时，协议继续：

(1)A 把密钥发给 B。

(2)B 解密信息展示位内容。他检查他的随机串来验证位的有效性。

如果信息不包含 B 的随机串，A 可偷偷地用各种密钥解密她递交给 B 的信息，找出与约定不同的位。因为每位仅有两种可能值，极易修改。B 的随机串防止她使用这种攻击，她必须修改新信息而且要正确地复制 B 的随机串。如果加密算法是好的，她找出的机会是极小的。在 A 约定后，她不能改变她的位。

使用单向函数的位约定协议

(1)A 产生两个随机位串，R_1 和 R_2。

R_1,R_2

(2)A 生成一个信息，它包含她的随机串和她期望约定的位(实际上为几位)。

(R_1,R_2,b)

(3)A 对信息作哈希运算，把结果以及一个随机串一起发给 B。

$H(R_1,R_2,b),R_1$

来自 A 的这一传送内容是约定的证据。A 在步骤(3)使用单向函数，使 B 不能函数求逆算出各位的值。

A 展示时，协议继续：

(4)A 把原始信息发给 B。

(R_1,R_2,b)

(5)B 对信息作哈希运算，此时的哈希值和 R_1 与步骤(3)收到的哈希值和 R_1 进行比较，如果匹配，位有效。

这个协议比以前的协议的优点是，在协议建立后仅需要单向通信：B 不能同 A 谈话。A 把约定的位的信息和另一个展示位的信息发给 B。

因为 A 约定的结果是用单向哈希函数操作信息，所以不需要 B 的随机串。A 不能欺骗和

找出另一个信息(R_1,R_2,b′),使得F(R_1',R_2',b′)=F(R_1,R_2,b′)。如果A不能把R_1发给B,她将改变R_2的值。B知道R_1的措施防止A做欺骗活动。

伪随机序列生成器的位约定协议

(1)B产生一个随机位串并发给A。

R_B

(2)A由伪随机位生成器产生随机数种子。对应B随机位串的每一位,她发给B的内容为:

(a)如果B的位是0,则生成器输出。

(b)如果B的位是1,则生成器的输出与她约定位的异或。

A展示时,协议继续:

(3)A把她的随机数种子发给B。

(4)B完成步骤(2)确认A的活动是公正的。

如果B的随机位串足够长,伪随机位生成器是不可预测的,A不可能实施欺骗活动。

斑点(Blobs)

A发给B约定的位串有时称为斑点。尽管协议没有理由作这样规定,斑点是一串位序列,正如Giues Brassard所说,"如果是有用的,证明他们是好的"。

斑点的特性为:

- A能对斑点作出约定。约定斑点,正是约定位。
- A能打开她已约定的任一斑点。当她打开斑点时,她能使B相信当她约定斑点时她约定位的值。她不能打开既为0又为1的任何斑点。
- B不知道A怎样能打开她约定的任何未打开的斑点。甚至在A打开其它斑点后,他还是不知道。
- 斑点不能生成与A约定位不同的任何信息。斑点本身以及A约定的和打开他们的过程是同任何事情不相关的,A可望对B保持秘密。

上面的所有协议都满足这四个特性。

4.7 公正掷币

通过投掷一枚硬币,以其正、反面来做一个二值决定(如是与非、出与留等)。这是一个十分普遍的西方习惯。假如一时没有一枚钱币,怎么来完成同一任务呢?下面的一些方法都可做到公正掷币。

使用位约定协议

(1)使用前节列出的任一个位约定方案,A约定一个随机位。

(2)B去猜测位的内容。

(3)A向B展示她的位内容。如果猜对,B为掷币游戏的赢家。

一般地说,我们需要的协议具有这些性质:

- A掷币在B猜测前。
- 在听到B的猜测后,A不能再掷币。
- 在猜测之前,B不能知道钱币怎样落地。

使用单向函数的掷币

如果A和B同意用单向函数,这个协议是简单的:

(1)A选一个随机数X。她计算Y=F(X),其中F(X)为单向函数。

(2)A 把 Y 发给 B。

(3)B 猜 X 是否为偶数或奇数，并把他的猜测发给 A。

(4)如果 B 的猜测是正确的，掷币的结果是正面。如果 B 的猜测不正确，掷币的结果是反面。A 宣布掷币的结果并把 X 发给 B。

(5)B 确认 $Y=F(X)$

这个协议的安全性依赖于 F(X)的选择。如果 A 能找出 X 和 X′，例如 X 为偶数而 X′为奇数和 $Y=F(X)=F(X')$，那末她每次都能欺骗 B。另外，F(X)必须等概率地产生偶数和奇数。否则，B 至少在某段时间能欺骗 A。例如，如果 F(X)在 75%的时间内产生偶数，B 在步骤(3)连续地猜测偶数，有 75%的机会赢得掷币。

另一个协议用公钥密码术或对称密码术进行操作，仅需要算法变换，即：

$$D_{K_1}(E_{K_2}(E_{K_1}(M)))=E_{K_2}(M)$$

一般地说，对称算法不具有这个特性，但对一些公钥算法是正确的(例如 RSA 公钥算法)。协议为：

(1)A 和 B 产生公钥/私钥对(或在理论上一个可互换的私钥)。

(2)A 产生两个信息，一个指钱币的正面而另一个指反面。这些信息将包含一些唯一的随机串，使她能以后在协议中验证。A 用她的密钥加密两个信息并把他们发给 B。

$$E_A(M_1), E_A(M_2)$$

(3)B 不能读出任一个信息，只是随机地选择一个。用他的密钥加密后把结果送回 A。

$$E_B(E_A(M))$$

(4)A 不能读出发回给她的信息，用她的密钥解密并把它发回给 B。

$$D_A(E_B(E_A(M)))=E_B(M_1)\text{或}$$

$$D_A(E_B(E_A(M)))=E_B(M_2)$$

(5)B 用他的密钥解密信息并展示掷币的结果。他把解密信息发给 A。

$$D_B(E_B(M_1))=M_1\text{ 或}$$

$$D_B(E_B(M_2))=M_2$$

(6)A 读掷币的结果并验证随机串是正确的。

(7)A 和 B 展示他们的密钥，结果能验证相互没有欺骗。

这个协议是自行约束的。每一方能立即检测出另一方的欺骗行为。协议完成后，不需要可信任的第三方参与实际的协议或裁决。让我们试行欺骗看看协议怎样处理：

如果 A 想欺骗并强制掷币结果为正面，她有三种可能影响结果的方法。首先，她在步骤(2)加密的是两个“正面”信息。在步骤(7)当 A 展示她的密钥时，B 会发现这个问题。第二，在步骤(4)她不正确地加密信息。然后，不能解密并迫使该信息为无用的信息。在步骤(5)B 可发现这个问题。第三，在步骤(6)她对信息的有效性说谎。当 A 不能证明信息无效时，在步骤(7)B 也可发现这个问题。当然，A 拒绝参与协议的任何一个步骤，B 将明显地知道 A 的欺骗企图。

如果 B 想欺骗并强制为“反面”，他的选择是十分有限的。在步骤(3)他不正确地加密信息，当 A 在步骤(6)看到最后的信息时，会发现这个问题。如若步骤(5)完成不当，结果也是无用。A 在步骤(6)发现这个问题。他宣称，由于在 A 部分有一些欺骗信息，他不能适当地完成步骤(5)，但这种欺骗形式将在步骤(7)被发现。最后，他在步骤(5)把“反面”信息发给 A，不考虑他加密的信息，但 A 立即能在步骤(6)检查证实的信息。

掷币入井

所有的协议中有趣地发现，双方不会同时知道掷币的结果。协议中的要点为，一方（第一个协议为A，第二个协议为B和第三个协议为A知道掷币结果，但不能改变它。他只能把结果延迟展示给另一方。这称为掷币入井。设想往一口井中掷币，A靠近井，B远离它。B扔钱币，钱币落在井中。A能看，但不能改变掷币结果。B只有在A让他靠近才能看到结果。

掷币密钥生成

这个协议的一个实际应用是密钥生成。掷币协议允许A和B生成一个随机的密钥，使得密钥互不影响。假设A和B加密他们交换的所有密钥，这种密钥生成安全地防止窃听。

4.8 心理扑克

类似公正掷币协议3，可建立一个允许A和B通过电子邮件相互玩扑克的协议。不同于生成和加密钱币正、反面两个信息，A生成52张扑克牌的信息，M_1、M_2、……M_{52}，每一信息对应一张牌（这与公正掷币中不存在物理钱币一样，心理扑克也不是实际的扑克牌）。B随机地选择五个信息，用他的公钥加密他们，把他们送回给A。A解密信息，并把他们送回给B，B解密后用来决定他手中的牌。然后，他再送出另5个的信息给A，A解密后用来决定她手中的牌。在游戏中，打牌双方重复使用另外的纸牌。游戏结束，A和B展示他们的纸牌和密钥，使得确信他们没有相互欺骗。

三人玩的心理扑克

多人玩扑克更有乐趣。基本心理扑克协议能容易地扩充至三个或更多的选手。此外，密码算法必须是互换的。

(1)A、B和C都生成一个私钥，或一对公开秘密的密钥。

(2)A产生52个信息，每一信息对应一张纸牌。这些随机信息含有一些唯一的随机串，这样她能在以后验证。A用她的密钥加密所有信息并把他们发给B。

$$E_A(M_n)$$

(3)B不能读任何信息，随机地选择五张纸牌。他用他的密钥加密并送回给A。

$$E_B(E_A(M_n))$$

(4)B把剩余47个信息发给C。

$$E_A(M_n)$$

(5)C不能读任何信息，随机选择五个。她用她的密钥加密并发给A。

$$E_C(E_A(M_n))$$

(6)A不能读返回给她的任何信息，用她的密钥解密并送回给B或C(取决于他们来自何处)。

$$D_A(E_B(E_A(M_n)))=E_B(M_n)$$

$$D_A(E_C(E_A(M_n)))=E_C(M_n)$$

(7)B和C用他们的密钥解密信息，展示他们手中的纸牌。

$$D_B(E_B(M_n))=M_n$$

$$D_C(E_C(M_n))=M_n$$

(8)C从剩余的42个之中随机地选择五个额外信息，发给A。

$$E_A(M_n)$$

(9)A解密信息，展示在她手上的纸牌。

$$D_A(E_A(M_n)) = M_n$$

(10)A、B 和 C 都展示他们手上的纸牌和他们的所有的密钥，使得每一个人能确信没有人进行欺骗。

另外的纸牌以相同的方法处理。如果 B 和 C 想要一张纸牌，他们取出一叠加密的纸牌并与 A 一起完成协议。如果 A 想要一张纸牌，无论当时谁有一叠纸牌，就随机发给她一张。

理论上无需步骤(10)。人们不需要在协议结束展示他们手上的牌；仅那些没有结束的选手需要展示他们手上的牌。步骤(10)仅是设计来发觉有否欺骗用的，也许这是多余的步骤。

在扑克中，关心的是赢者是否进行了欺骗。输的人是否欺骗并不重要。因此，让我们看不同选手赢的情况。

如果 A 赢了，她展示她手上的牌和她的密钥。B 能使用 A 的私钥确认 A 正确地完成了步骤(2)。C 用 A 的密钥加密纸牌并验证他们与步骤(8)她发给她的加密信息相同的方法可确认 A 没有对她手上的纸牌说谎。

如果 B 或 C 赢了，他们展示他们手上的牌和密钥。A 检查她的随机串确认纸牌是合法的。她也能确认纸牌是用赢者的密钥加密的纸牌，并验证他们与在步骤(3)或(5)她接收的加密信息相同。

如果有人发现其他人进行欺骗的证据，这了捕捉罪犯，选手必须展示他们的所有信息。但此时游戏已友好地停止了；展示手上的纸牌变得不再重要。

这个协议不能安全地防止怀有恶意的选手之间的勾结。A 和另一个选者者能有效地联合起来对付第三个选手，共同骗出那个选手所做的事情而没引起怀疑。因此，重要的是检查所有的密钥和每次玩牌者展示在他们手上的随机串。如果你与其他二人坐在虚拟的桌旁而他们从不展示他们手上的纸牌，无论谁是发牌者，永不展示他们手上的纸牌(上面协议中的 A)，则停止玩牌。

扑克协议的攻击

如果使用 RSA 公钥算法，密码学者已发现这些扑克协议中会有很少量的信息泄露。特别地，如果纸牌的二进制表示为二次余数，纸牌的加密也是二次余数的。这个特性能用来“标志”一些纸牌。例如，所有的“A”牌，虽不能展示更多手中的牌，但在游戏中，对长时间的打牌来讲，任何微小的信息都会占上风。

有人发展了二人扑克协议修补了泄露问题，但十分复杂，还没有人研究消除信息泄露问题的 n 人扑克协议。

匿名密钥分配

考虑一下密钥分配的问题。如果我们假设人们不能产生他们自己的密钥(他们必须为确定的形式、或必须由一些组织签名，或类似的事情)，那么我们必须建立一些中心密钥服务机构，产生密钥并分配密钥。问题是我们必须解决分配密钥的方法，做到无人(包括服务器)能知悉或想象出谁得到什么密钥。

协议如下：

(1)A 产生公钥/私钥对，在这个协议中，她有两种密钥的秘密。

(2)KDC 产生连续的密钥流。

(3)KDC 用它自己的私钥，一对一地加密密钥。

(4)在网络上，KDC 一对一地传送加密的密钥。

(5)A 随机地选择一个密钥。

(6)A 用她的密钥加密选择的密钥。

(7)A 等待一会(足够长使服务器无法知道她选择的密钥),把双倍加密密钥送回给 KDC。

(8)KDC 用它的私钥解密双倍加密密钥,留下用 A 的公钥加密的密钥。

(9)服务器把加密的密钥送回给 A。

(10)A 用她的私钥解开密钥。

位于这个协议中间的窃听者 E,无法知道 A 选择什么密钥。她看到步骤(4)连续通过的密钥流。当 A 在步骤(7)把密钥送回给服务器时,它被她的公钥加密,其中公钥也是秘密的。窃听者无法与密钥流相联系。当服务者在步骤(9)把密钥送回给 A 时,它也被 A 的公钥加密。仅当在步骤(10)A 解密密钥时,展示出密钥。

如果使用 RSA,这个协议具有泄露的信息,速率为每个信息一位。它再次是二次余数的。如果你继续按这种方法分配密钥,需要弄清楚该泄露是否无关要紧。当然,如果 A 不信任 KDC,泄露是不可避免的。不怀好意的 KDC 估计可能保留它产生的每一个密钥的记录,用来检索后决定 A 的信息。

本协议也假设 A 继续公正地活动。使用 RAS,她能做许多事,得到比使用其他方法得到更多的信息。但在目前的情况这不是一个问题。后面的协议将讨论这个问题。

第5章 高级协议

5.1 公正的密码系统

法庭授权的侦听电话是目前社会中最有效的防止犯罪行为并将罪犯绳之以法的方法之一。从法律上考虑，公钥密码术的广泛使用可能对犯罪和恐怖组织的活动提供方便。许多提案建议政府中的合适部门，在法律允许范围，应能获得公共网络上的任何通信的明文。目前，需要强制市民：(1)使用弱密码系统，即合适的当局无需作极大的努力就能破译。或(2)最好将他们的私钥交给当局。毫不奇怪，这些措施告诫人们要正视不快的感受，隐私会面临与国家安全和法律实施的冲突。

因此研究一种既保护个人的秘密同时又允许法庭授权侦听的密码系统是富有挑战性的。

Micali 提出了公正的密码系统。在这种密码系统中，私钥被分成几片并分配给不同的当事人。像秘密共享方案那样，有关人员把他们放在一起才能构成私钥。每片都附有其它性能，用于重建密钥之前对每片作正确性检查。

按此 A 生成她自己的私钥，把每片分给几个保管人。任一保管人都不能恢复 A 的私钥。然而所有保管人均能验证他们所有的片是私钥的有效的片；如果法庭授权侦听，有关法律条文要求五个保管者交出他们的片。具有所有的五片，执法者再构造私钥，窃听 A 的通信。换句话说，为了能再构造 A 的密钥和使其有效，攻击者必须贿赂所有的保管者。

协议如下进行：

(1)A 产生她的公钥/私钥对。她把私钥分解为几个公共片和秘密片。

(2)A 把公共片和相应的秘密片发给每一个保管者。这些信息必须加密。她也把公钥发给 KDC。

(3)保管者完成计算确认公开片和秘密片是正确的。每一人保管者把秘密片存于某个安全的地方并把公开片发给 KDC。

(4)KDC 完成公开片和公钥的另一项计算。假设一切都正确，它签署公钥，并把它送回给 A 或放入某一数据库中。

如果法庭(或另一方)要求侦听，每一个保管者应交出所保管的片给 KDC，KDC 能再建私钥。KDC 或任一保管者不能再建私钥；再建密钥需要所有的保管者在场。

任何公钥密码术的算法能以这种方法公正地执行。Micali 讨论了把它与门限组合在一起，所以有保管员的子集(例如，五中取三)可再建私钥。他也指出可把它与茫然传送结合在一起，使得保管者不知道谁的私钥正在被建。

5.2 或全或无的秘密揭示

假设 A 为某国的间谍，现被解雇。为了赚钱，A 出卖秘密。任何愿出钱的人都能买到秘密。她将所有秘密顺序列表形成一个目录，每一密钥都有着诱人的标题。

A 不会以一个秘密的价格卖出二个秘密，甚至不愿在一个秘密外另加一点其它秘密的部

分信息。一个有购买潜力的买主，不想为随意的秘密付钱。他也不想告诉 A 他想要什么秘密。因为这与 A 无关，此外，也避免 A 可能在她的秘密目录中加上“B 对某某秘密感兴趣”一条作为新内容。解决的办法称为或全或无的秘密揭示(ANDOS)，因为，只要 B 得到 A 一个秘密的任何信息，他就失去了解其它秘密的机会。

扑克协议不能以这种方式工作。因为在协议结束，A 和 B 必须相互展示他们手中的牌。此外 B 有许多技巧得知多个秘密。

5.3 秘而不宣的知情证明

通常情况甲向乙证明他知晓某事就是把该事告诉乙，这不是一个好办法。因为甲一旦说出后，就再也不能控制信息的扩散，乙完全可能传播出去而使之完全暴露。甲怎样既能证明他确实知晓某事，而又不泄漏事情本身？

这可用单向函数来实现。甲一点音信也不给乙就证明了他确实知晓某事，这称为秘而不宣的知情证明。

秘而不宣的知情证明协议可用岩洞的故事来解释。岩洞如图 5.1 所示，其奥妙是只有知道作出知情咒语的人能打开 C 和 D 之间的暗门。此外对任何人两个通道都是死路。

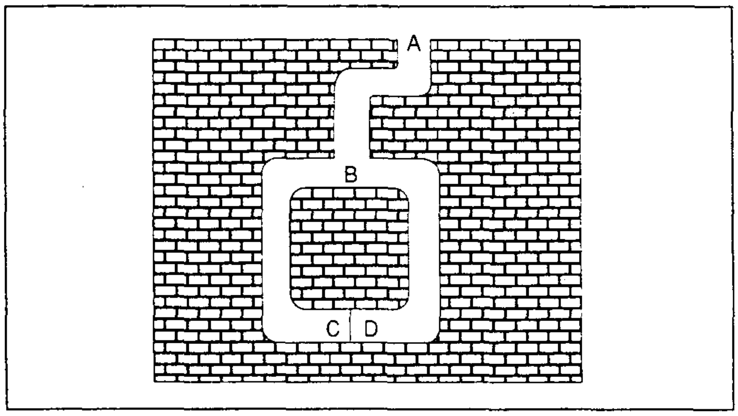


图 5.1 用于秘而不宣知情证明示意的岩洞

甲知道岩洞的秘密，他想向乙作出知情证明，但她不宣示这个咒语。步骤如下：

(1)乙站在 A 点。

(2)甲走进岩洞，或到 C 或到 D 点。

(3)在甲进入岩洞消失后，乙走到 B 点。

(4)乙叫甲并向她提出从左通道出来或从右通道出来的任一要求。

(5)甲应答，如果需要的话使用魔语打开暗门。并根据当时所处位置的需要用咒语打开暗门。

(6)甲和乙重复步骤(1)至(5)达 n 次。

这种技术称为分割选择技术。分割选择使甲无法反复地猜出乙要求她从哪一边出来。如果甲不知道秘密，她将从她进去的路出来，而不能从另一边出来。但她有 50%的机会猜测乙会选择左和右的那一边，使得她有 50%的欺骗机会。两个循环欺骗机会是 25%，n 次后，欺骗的机会为 2^n 分之一。16 次循环后，甲每次欺骗乙的机会为 65,536 分之一。乙对甲所做的 16 次证明都是正确的，毫无疑问甲知道打开 C 和 D 之间暗门的咒语。

用岩洞故事模拟秘而不宣的知情证明并非完善。因为它十分简单，只要甲能从一边进去而从另一边出来就可证明；并不一定要作任何分割选择。但是严格的数学证明需要分割选择协议。

假设甲知道一条信息，它是某难题的解。基本协议就应包括几个循环。

(1)甲使用她的信息和一些随机数把难题转换为另一个难题，它与与原始问题同态。她使用她的信息和随机数解新的难题。

(2)甲使用位约定方案，提交新难题的解。

(3)甲向乙揭示新的难题。乙不能由新题获得关于原题或其解的任何信息。

(4)乙问甲下列问题之一：

(a)向他证明新老问题是同态的，或

(b)公开并证明步骤(2)提交的解是新问题的解。

(5)甲同意并执行之。

(6)甲和乙重复步骤(1)至(5)达 n 次。

这类数学证明是复杂的。必须仔细地选择问题和随机变换，使得乙即使协议多次反复执行，也不能获得关于原始问题的任何信息。当然并非所有"难题"都能用秘而不宣的知情证明，但大部分能这样做。

哈密尔顿回路可作为一个例子来解释以上概念，它来自图论，由 Mannel Blum 第一个提出。由各点连线组成的网络称为图。A 知道一个回路，沿线连续途径各点且仅通过各点一次，这称为哈密尔顿回路。

对一个非常大的图，找这条回路需花几年的计算时间；它是以前讨论的单向函数。若甲知道这一信息，她要乙相信但又不给出解。

甲知道图 G 的哈密尔顿回路。协议如下工作：

(1)甲随机地置换 G。她移动图上的点，改变他们的坐标产生一个新的图 H。因为 G 和 H 拓扑上为同态的(即相同的图)，如果她知道 G 的哈密尔顿回路，那未她能容易地找出 H 的哈密尔顿回路。如果她不是自己产生 H，决定两者之间的同态将是另一个难题，也要花几年计算时间。

(2)甲给乙一个 H 的拷贝。

(3)乙问甲下列问题之一：

(a)向他证明 G 和 H 是同态的，或

(b)向他显示 H 的哈密尔顿回路。

(4)甲应答下列之一：

(a)证明 G 和 H 是同态的，不显示 G 或 H 的哈密尔顿回路，或

(b)显示 H 的哈密尔顿回路，不证明 G 和 H 拓扑上为同态的。

(5)甲和乙重复步骤(1)至(4)达 n 次。

如果甲是诚实的，她能为乙提供步骤(3)两者之一的证明。然而，如果她不知道 G 的哈密尔顿回路，她不能产生满足两个需求的 H。她能做的最好的方法是产生一个回路，与 G 同态或有相同的点和线及一个有效的哈密尔顿回路的回路。而她有 50%的猜测机会。乙要求她在步骤(3)完成那一个证明，乙重复协议足够多使他相信甲知道 G 的哈密顿回路。

因为乙无法获得任何信息帮助他决定原始图的哈密尔顿回路，这是秘而不宣知情证明协议。每次循环，甲向他显示两者之一。她向他显示两个图为同态的。因为找出新图的哈密顿回

路就像找出老图的哈密顿回路一样困难,所以这不能帮助乙。换句话说,她向他显示新图的哈密顿回路。这也不能帮助他,因为找出老图和新图之间同态性比找出两者之一的哈密顿回路更难。因为甲每次循环置换不同的原始图,无论协议进行多少次,乙得不到甲拥有的原始信息。

另一个例子是图的同态性。假设甲知道两个图 G_1 和 G_2 为同态的。下面的协议将使乙相信甲的知识:

(1)甲随机地置换 G_1 的顶点,并产生另一个图 H,与 G_1 同态。因为甲知道 G_1 和 H 之间的同态性,所以她也知道 H 和 G_2 之间的同态性。对另一些人,找出 G_1 和 H 或 H 和 G_2 之间的同态性就像找出 G_1 和 G_2 之间的同态性一样难。

(2)甲把 H 送给乙。

(3)乙问甲下列问题之一:

(a)证明 G_1 和 H 是同态的,或

(b)证明 G_2 和 H 是同态的。

(4)甲应答下列之一:

(a)证明 G_1 和 H 是同态的,不显示 G_2 和 H 是同态的,或

(b)证明 G_2 和 H 是同态的,不证明 G_1 和 H 是同态的。

(5)甲和乙重复步骤(1)至(4)达 n 次。

如果甲不知道 G_1 与 G_2 之间的同态性,她不能产生与两者同态的 H。她产生一个图与 G_1 为同态的或与 G_2 为同态的。像前面的例子,她有 50%的机会猜测乙要她完成步骤(3)的那一个证明。

协议没提供乙任何有用的信息,帮助他构画出 G_1 和 G_2 之间的同态性。因为甲对协议的每次循环产生一个新图 H,即使多次循环后,乙还是不能获得任何信息。他不能从甲的回答中构画出 G_1 和 G_2 之间的同态性。

并行秘而不宣的知情证明

上述基本的协议包含甲和乙之间的 n 次交换。为何不把 n 次交换作并行处理呢?并行证明可如下执行:

(1)甲使用她的信息和一些随机数把难题转成为 n 个不同的同态的问题,她使用她的信息和随机数解决 n 个新的难题。

(2)甲提交 n 个新的难题的解。

(3)甲向乙揭示 n 个新的难题。乙不能使用新的问题获得关于原始问题或它的解的任何信息。

(4)对 n 个新的难题的每一个,乙问甲下列问题之一:

(a)向他证明旧的和新的问题是同态的,或

(b)在步骤(2)公开她提交的解,证明它是新的问题的解。

(5)甲应答 n 个新的难题的每一个。

这个协议是安全的,因为甲、乙间传递信息的次数较少(尽管信息较长)。

能使第三方相信吗?乙想使丙相信甲知道信息。他保留协议的副本,并显示副本给丙。丙能相信吗?

不,因为不知信息的两个人能串通起来欺骗他。他们能事先约定回答乙在某一个循环将问的问题。据此,攻击者不知道秘密,但可装成知道。他与乙串通,让乙总是问他能回答的问题。乙和攻击者之间协议的副本,与乙和诚实的甲之间协议执行的结果。对丙来说无法分析出不

同。

非交互式秘而不宣的知情证明

由于以上协议是交互的，丙没有参加交互他不会相信，为了使丙和感兴趣的任何人相信，我们需要非交互的协议。

甲公布证据，任何人只要化时间去核对这些证据都将证明是确凿的。

该协议与并行协议类似，但单向函数取代了乙：

(1)甲使用她的信息和随机数把难题转换为几个不同的同态问题。她使用她的信息和随机数解几个新问题。

(2)甲提交几个新问题的解。

(3)甲使用所有约定作为单向哈希函数的输入。她保存这个单向哈希函数的输出的开始 n 位。

(4)甲取出在步骤(3)产生的 n 位；根据第 n 位是 0 还是 1，她取第 n 个新的难题和下列之一：

  (a)证明旧的和新的问题是同态的，或

  (b)公开步骤(2)她提交的解，证明它是新问题的解。

(5)甲公布步骤(2)以及步骤(4)的结果。

(6)乙、丙或任何感兴趣的人，检查从步骤(1)至步骤(5)是否正确地执行。

惊人的是：甲公布的信息并不包含她的秘密信息，但能使任何人相信她拥有秘密。

协议成立的原因是单向哈希函数起着一个非基本的随机位生成器的作用。如甲想进行欺骗，他必须能预测单向哈希函数的输出(记住，如果她不知道难题的解，她能做步骤(3)的(a)或(b)，但不能做两者)。然而，甲无法知道单向函数产生的确定位，或猜测哪一位将产生。事实上单向函数在协议中代替了乙：随机地在步骤(3)中选择两个证明之一。

在非交互协议中，有许多需求/应答序列的重复。甲用随机数选出难题；而乙、丙等验证人无法控制，因此甲能挑出不同的问题，不同的约定值直至哈希函数产生他所喜欢的，因此非交换协议要求更多的重复次数，64 次、128 次，以减少甲进行欺骗的概率。

概括

Blum 证明任何数学理论能被转变成图，使得理论证明等有效为哈密尔顿回路。用这种方法，任何数学证明能被转变为秘而不宣的知情证明。使用这种技术，研究者能证明他们知道某一原理的解而不揭示解是什么。实际上，Blum 就可以这样来发布他的研究成果。

几年来，无论在理论上或应用上都已完成了许多工作。

5.4 用秘而不宣的方法证明身份

在现实世界中，我们使用物理标志作身份的证明：护照和身份证、驾驶证、信用卡等。这些卡片中含有与当事人有关的一些东西：通常如照片，同样也可用指纹，视网膜扫描，牙齿的 X 射线照片。用数字化技术来做同样的事不也是很好吗？

用秘而不宣的方法证明身份首先由 Vriel Feiget 等提出。A 的私钥可证明她身份的功能。使用秘而不宣的方法，证明她知道她的私钥，进而证明她的身份。

这个想法是相当有用的。使人们可以证明他们自己的身份而不用任何物理标志。然而并不完善，可举出一些实例。

象棋大师问题

甲不知道象棋规则，可她能击败国际大师。在同时同地但不同房间，她向 A、B 两位国际级

大师发起挑战。她用白棋对A，黑棋对B。国际大师相互不知道。B走第一步白棋。甲记录下棋谱并走进与A对弈的房间。她执白棋，对A走相同的棋，而A走每一步黑棋，甲记录下棋谱并走进与B对弈的房间，走相同的棋。如此继续直到赢一场和输一场(或两场平局)。

实际上，是A正同B下棋，甲简单地扮作一个中间人，在另一张台子上模仿一个大师走棋。然而，如果A或B不知道双方的存在，每一个将十分吃力地以为与甲对弈。

黑手党的欺骗行为

A在黑手党开的餐馆中进餐。B为其服务，C正在D的商业中心，一个昂贵的珠宝商店购物。B和C是黑手党的两个员工，有秘密的无线通信联系。A和D没意识到欺骗行为。

A就餐结束，当她准备付钱并向B证明她身份时，B通信告之C准备进行欺骗行为。C买一些昂贵的钻石，也准备向D证明她的身份。现在，当A向B证明她身份时，B借无线通信告知C，C与D完成相同的协议。当D问问题时，C用无线通信把问题返回给B，B问A。得到A回答后，B将正确答案经无线通信告诉给C。实际上，A正在向D证明她的身份，B和C位于往返交换信息之间。当协议完成后，A已向D证明了她的身份并被认为是买主，而C骗得昂贵的钻石后消失了。

恐怖分子的欺骗

如果A愿意同C协作，他们也能欺骗D。在这个协议中，C是一个著名的恐怖分子。A帮助她进入国境。D是移民局官员。A和C通过秘密无线通道联系。

当D问C证明身份的部分问题时，C用无线通信送给A，让A回答他们的问题。C收到答案回答D问题。实际上，A向D证明了她的身份，D认为C是A，让他进入国境。三天后，C实施了一起恐怖活动。

以上二例欺骗是可能的，因为共谋者能通过秘密电台通信。防止它的一种方法是将整个验证过程置于抗电磁辐射的屏蔽网中。由于切断通信而不能欺骗。

Thomas Beth 和 Yuo Desmedt 提出了另一个解决办法，使用准确的时钟。如果协议的每一步必须发生在给定的时间间隙中，那么将没有时间用来同协作者通信。以象棋大师故事为例，如果每一步棋计时一分钟，A将没有时间从一个房间到另一个房间。

移动身份的问题

还有一些破坏协议的例子。

多身份欺骗：在一些应用中，登记公共密钥时，不加检查，因此，A可得到几个私有密钥即有了几个身份。这可便于她逃税。A也能犯罪后消失。首先，她生成和公布几个身份。其中之一从不使用。然而她偶尔一次去犯罪，然后，立即停止使用。证人知道犯罪者的身份，但因A从不再使用那个身份，她是难以跟踪的。

为了防止这种情况，必须有一些机制使每一个人仅有一个身份。例如建议每个婴儿出生时只有一个身份。

租用护照：A想去某国旅游，但政府不发给她签证。C把她身份租给A。C把她的私钥卖给A，A以C身份进入该国的国境。

C不但收取了买她身份的钱，而且有了十分好的犯罪不在现场的证据。当A在该国时，她可进行犯罪。“C”证明她在该国，怎样可能是她犯罪呢？

当然，A也能栽赃。在她离开之前或返回之后，她以C的名义犯罪，证实自已为C(她有C的私钥，使她能容易做到)，然后犯罪并逃离。警察调查C。

以上各例都说明了不采用直接真正地证明身份的方法存在着的弊端，科学和技术不能完

全取代法律和解决社会问题，身份证和护照还是十分重要的，必须好好保存。

5.5 盲签名

所有数字签名协议的基本点是文件的签名者总是知道他们正签署的内容。

有时我们想让人签字而不让他们看文件的内容。它虽非常有用，但后面将用它实现投票和数字支票协议。有许多方法使签名者几乎知道他们签什么。

完全地盲签名

B是一个公证人。A要他签署文件但不让他对签署的内容有任何了解。B不关心文件的内容；他仅证实某时该文件作了公证。他情愿做这种事。

(1)A取文件并乘以一个随机值。这个随机值称为盲因子。

(2)A把盲文件发给B。

(3)B签署盲文件。

(4)A除去盲因子，保留B签名的原始文件。

如果签名功能和相乘功能是互易的，协议可行。如果不是，不能用乘而用其它方法修改文件。在此，假设操作是相乘和所有数学运算。

B能进行欺骗吗？他能收集到签署的文件的任何信息吗？如果盲因子真正是随机的，他不能。B要签的在步骤(2)的盲文件一点也不像A开始的文件。B在步骤(3)签好的盲文件也不像步骤(4)所得的签名文件。即使B文件拿到手，完成协议后，他不能证明(对他或任何人)他在这个文件上签名。只知他的签名是有效的。知道他(或有人也用他的私钥)签了名；他能像其他任何人那样，验证签名。他无法收集签名文件的任何信息。即使签名一百次，他仍无法知道签名文件的内容。

完全的盲签名特性为：

· B签名的文件是有效的。签名证明B签了名。如果向B显示，他将确认他签名的文件。同时完全的盲签名也有前面讨论的数字签名的所有特性。

· 签名者不能收集在签名文件活动中的签名文件。即使他记录每次盲签名，当出现一个签名文件时，当他签名时无法确定它的内容。

用盲签名协议，A可让B对各种内容签名；甚至如“B欠A100万美元”等，由于签名内容毫无限制，无需说，这个协议并不太实用。

盲化了的文件像是在信封中的文件，文件的盲化过程就是把文件放入信封，去除盲因子则像打开信封。盲签名就如有一张复印纸在信封和文件之间，签名者只在信封上签字，而通过复印纸签在文件上，他不知道文件的内容。

第6章 深奥协议

6.1 茫然传送

密码破译者 B 努力地做一个 500bit 数字的因子分解，除知道它由 5 个 100bit 数字产生外其它什么都不知道。A 偶然得知其中的一个因子，于是她将以 100 元的价格出售给 B。B 对此感兴趣，但他仅有 50 元。于是 A 提出将 50 比特卖给 B，虽不完整但可节约 B 大量的工作。但是 B 需要找出一个方法能确定这 50 位是该因子的一部分后才会买。

显然不可能指望 A 会显示该因子，这里需要引入茫然传送的概念。A 传送一组信息给 B。B 收到那组信息的一个子集，A 不知道他收哪一个子集。然而这还没有完全地解决上述问题。B 随机地收到 50 比特后，A 必须用秘而不宣的知情证明使他相信这些比特可构成因子。

在下面协议中，A 仅能将两条消息中的一条有效地发给 B。B 收到一条消息，A 不知道哪一条。

(1)A 产生两个公钥的密钥对，或四个密钥。她发给 B 两个公钥。

(2)B 按对称算法(例如 DES)选择密钥。他随机挑选一个 A 的公钥并用它加密他的 DES 密钥。他把加密的密钥发给 A，但不告诉使用 A 的哪一个公钥进行加密。

(3)A 用她的两个私钥解密 B 的密钥。一种情况，她使用正确的密钥，成功地解密 B 的 DES 密钥。另一种情况，她使用错误的密钥，仅能产生一串无意义的比特，尽管看似像一个随机的 DEC 密钥。因为她不知道正确的明文，她无法知道哪一个正确。

(4)A 用前一步生成的每一个 DES 密钥(一个真正和一个无意义的)加密消息，并把两条消息发送 B(有效地发送消息只一条)。

(5)B 企图解密 A 的两条消息，但仅能成功地解密其中之一。这就是一个完整的茫然传送过程。B 已收到了他要的消息，A 无法知道两条中的哪一条。

(6)协议完成和知道传送的两种可能结果后，A 必须把她的密钥给 B，让 B 验证她没有欺骗。毕竟，她在步骤(4)用两个密钥加密过同一消息。

协议安全地防止了 A 的攻击，因为她无法知道两个 DES 密钥哪一个为真正的密钥。协议安全地防止了 B 的攻击，因为在协议完成之前，他无法得到 A 私钥以确定用哪一个 DES 密钥加密消息。这个协议仍可看成是一个通过 Modem 掷币的协议，只不过更为复杂而已。

茫然传送协议也有非交互的，不经相互通信而实现。

本章许多其它的协议利用了茫然传送协议。

6.2 同步的合同签名

具有仲裁者的合同签名

A 和 B 想签一份合同。他们已同意文字叙述，但双方都不想先签。面对面时，两人可同时签名。远距离的话，他们只能借助于仲裁者。

(1)A 签文件的副本并发给仲裁者。

(2)B 签文件的副本并发给仲裁者。

(3)仲裁者发一条信息给 A 和 B,告诉另一方已在合同上签名。

(4)A 签合同的两个副本并把它们发给 B。

(5)B 签合同的两个副本,自己保留一份,另一份发给 A。

(6)A 和 B 通知仲裁者,他们每人有一份他俩签名的合同。

(7)仲裁者撕掉仅有一人签名的两份合同的副本。

由仲裁者预防了任何一方的欺骗,所以协议正常操作。如果 B 拒绝在步骤(5)在合同上签名,A 可向仲裁者要一份 B 已签名的合同的副本。如果 A 拒绝在步骤(4)在合同上签名,B 可同样行事。当仲裁者告诉他已收到步骤(3)的两份合同,A 和 B 知道他们因合同约束在一起。如果仲裁者没收到双方的合同,他撕掉已收到的一份合同,于是双方无约束。

无仲裁者(面对面)的同步的合同签名

如果 A 和 B 面对面相座,他们用这种方法签合同:

(1)A 签她名字的第一个字母并把合同递给 B;

(2)B 签他名字的第一个字母并把合同递给 A;

(3)A 签她名字的第二个字母并把合同递给 B;

(4)B 签他名字的第二个字母并把合同递给 A;

(5)一直继续直到 A 和 B 签完他们完整的名字。

如果你不理会双方签名字母长短不一,协议正常进行。虽然仅签下第一个字母并不能使合同法律生效,但双方已建立了信任,一般总会完成签约。因为双方都是想要签署合同的,仅仅不想在对方之前签名罢了。

无仲裁者(非面对面)合同的同时签名

在这个协议中,A 和 B 交换一系列的书面签名信息:我以概率 P 同意,并受此合同约束。

该信息的收据交给法官,法官将知道签约人对合同承担的概率 P。

(1)A 和 B 同意在某时期完成协议签名。

(2)A 和 B 决定他们愿意接受的概率差。如,A 愿以不高于 B2%的概率受合同约束。称 A 的差值为 a;称 B 的差值为 b。

(3)A 把以 P=a 签名信息发给 B。

(4)B 把以 P=a+b 签名信息发给 A。

(5)让 P′为 A 在前一步从 B 收到的信息的概率。A 以概率 P=P′+a 或加 1%,或任一较小值把签名信息发给 B。

(6)让 P′表示 B 在前一步收到 A 信息的概率,B 以 P=P′+b 或加 1%,或任一较小值把签名信息发给 A。

(7)A 和 B 继续交换步骤(5)和(6)直到以概率 P=1 收到信息,或直到超过步骤(1)同意的日期。

随着协议的进行,A 和 B 认为以越来越大的概率受合同的约束。例如,A 的第一个信息表示她以 2%的概率受约束;B 可能以 3%概率受约束。A 的下一个信息以 4%的概率受约束,等等。直到两者以 100%的概率受约束为止。

如果 A 和 B 完成协议,这是正常操作。如果协议完成的日期超出,一方伴随另一方的最后签名信息以裁决方式执行合同。裁决简单地选择一个 0 与 1 之间的随机值。如果随机值小于另一方签名的概率,双方受合同约束。如果随机值大于概率,双方不受约束。(裁决保留此值,

在认为相同的合同的事件中起作用。)这就是所谓以概率 P 受合同约束指的意义。

以上是基本协议，还有更复杂的协议。如果 A 以越来越小的增加概率开始签名，B 以同样方式响应。在缺少一方时，裁决将会干预。裁决作用于双方：约束或不约束。不会一方受约束而另一方不受约束。只要一方受约束的概率(无论如何小)稍有高于另一方，协议将终止。

没有仲裁者(使用密码术)合同的同时签名

密码学协议使用同样的小步逼近方法。尽管可用任何对称算法，但在协议描述中，使用 DES 方法。

(1)A 和 B 随机选择 200 个 DES 密钥，两两成对。这种密钥对并无特殊之处，只不过按协议要求组合起来。

(2)A 和 B 双方生成 100 对信息。例如，L_n 和 R_n，分别为“这是我第 n 个签名的左半部分”和“这是我第 n 个签名的右半部分”。标识符 n，从 1 变到 100。这些信息中可能也包括了合同的数字签名，以及前面定义的时戳。如果对方能生成某一对签名的 L_n 和 R_n，则考虑合同签名。

(3)A 和 B 按每一个 DES 密钥加密他们的信息，左边信息用密钥对的左边密钥，右边信息用密钥对的右边密钥。

(4)A 和 B 相互发送他们的 200 个加密信息，确保对方知道信息是哪一对的一半。

(5)A 和 B 相互发送每一个密钥对，使用茫然传送协议。即，A 独立发给 B 用于加密左边信息的密钥或用于加密右边信息的密钥，共计 100 对。B 做同样的事。交替发送每一半或一个先发送 100 对，然后另一个再发送 100 对，这无关紧要。现在，A 和 B 有每对密钥中的一个，但不知道对方有哪一半。

(6)A 和 B 使用他们收到的密钥解密半信息。并确信解密信息有效。

(7)A 和 B 相互发送所有 200 个 DES 密钥的第一个比特。

(8)对 200 个 DES 密钥的第 2 个比特，第 3 个比特，等等，A 和 B 重复步骤(7)直到 DES 密钥的所有比特传送结束。

(9)A 和 B 解密余下的半信息，签好合同。

为什么 A 和 B 必须经过所有过程？假设 A 想进行欺骗，看看发生什么事情。在步骤(4)和(5)，A 发给 B 无意义的比特串而中止协议。当 B 试图用他收到的密钥解密信息时，他能在步骤(6)发现有问题。B 可以安全地停止，因为 A 不能解密 B 的信息对。如果 A 是非常聪明的，她仅中止协议的一半。她正确地发送每对密钥的左边一半，但右边一半发送为无用的字符串。B 仅有 50%的机会收到右边的一半，使 A 在一半的时间内能进行欺骗。然而，这只是一对密钥的情况，如果有二对密钥，成功的机会为 25%。这就是为什么在协议中有 100 对密钥，A 必须正确地猜出 100 个茫然传送协议的结果。因为 A 仅有 2^{100}分之一的机会进行欺骗，如果 B 在步骤(6)没发觉问题，他可安然地认为没有欺骗。

A 也可在步骤(8)发给 B 任意比特。在收到整个密钥并试图解密一半信息时，B 发觉了问题。将立即知道 A 企图欺骗他。

或许 A 在得到足以破译 DES 的位数后停止发送正确的比特，例如 DES 密钥长度为 56 位，在收到 40 位后，只需 65，356 次尝试就可读取信息，这显然是一般计算能力所许可的，但 B 也能同样进行，他也有 40 位或 39 位 A 的密钥，所以 A 别无选择，继续执行本协议，

因为愚弄 B 的机会是非常小的，A 必须公正地操作。在协议结束，双方有 100 个签名信息对，其中任何一个足以满足有效签名。

这类协议有三个缺点。首先，如果一方的计算能力远远强于另一方，则有问题。例如，A 破

译算法的速度快于B，她能在步骤(8)首先停止发送比特并构造出B的密钥。B不能在适当的时间内做出同样的处理。

第二，如果一方先停止协议，则有问题。其困难是中止双方约束成单方约束，没有确切的期限。

第三，一些基本的数学假设产生的问题，如，有否使用成功的密码破译算法，使一方优于另一方。

6.3 数字认证的邮件

用于合同签名的同时茫然传送协议一样也能用于计算机认证的邮件，但需略作修改。假设A发一个信息给B，但要求没有签名收据不让他读邮件。现实生活中邮递员可处理此事，密码术也能做相同的事。

初看起来，同时合同签名协议能处理这些问题。A简单地用DES密钥加密她的信息。她的一半的协议像似："DES密钥左边一半为：32F5"。B的一边像似："这是我的收据的左半部分"。其他每件事都与前述的一样。

看看这为什么不能工作，要知协议依靠在步骤(5)茫然传送时双方保持诚实。他们知道发送给另一方的是有效信息的一半，但不知道哪一半。进入步骤(8)，由于摆脱的机会极小，他们也不能进行欺骗。如果A发给B的不是收据而是半个DES密钥，B在步骤(6)无法检查DES密钥的有效性，A仍能检查B收据的有效性，迫使B诚实。而A能方便地发给B一些无用的DES密钥，他不知道真假密钥的差异一直到A得到有效的收据。B何其不幸。

解决这个问题需要对协议做一些调整：

(1)A使用随机DES密钥加密她的信息并发给B。

(2)A产生100对DES密钥。每对的第一个密钥随机产生；每对的第二个密钥由第一个密钥和信息加密密钥异或产生。

(3)A用200个密钥中的每一个加密一个虚拟信息。

(4)A将整堆加密信息送给B，确保他知道每一信息是哪一对的哪一半。

(5)B产生100对随机的DES密钥。

(6)B产生一对作为有效收据的信息。最好选用"这是我的收据的左半部分"和"这是我的收据的右半部分"并带有某种可证实信息。他产生100对收据并编号。象前面的协议所述，如果A生成收据的两半(相同编号)和所有加密密钥的话，收据是有效的。

(7)B用DES密钥对加密每一个信息对，第n个信息对用第n个密钥对，每对的左边信息用左边密钥，右边信息用右边密钥。

(8)B发送信息对给A，确保A知道信息是哪一对的一半。

(9)A和B使用茫然传送协议相互发送每对密钥。即，对于100对的任一个，A发给B用于加密左边信息的密钥或用于加密右边信息的密钥。B做相同的事。他们能交换发送的一半或者先发送100中的一个，再发送其余99个是无关紧要的。目前，A和B有每对密钥的一个，但不知道对方有哪一半密钥。

(10)A和B解密他们的信息，并确保解密信息有效。

(11)A和B相互发送200个DES密钥的第一位(如果担心E能读邮件信息的话，他们相互加密传送的信息)。

(12)A和B处理重复步骤(11)200个DES密钥的第二位、每三位、等等，直到所有的DES

密钥的所有比特传送结束。

(13)A 和 B 解密信息对余下的一半。A 有一个来自 B 的有效收据,B 异或任一对密钥获得原始的信息加密密钥。

步骤(5)到(8)对 B,步骤(9)到(12)对 A 和 B,是和合同签名协议相同。新的诀窍是 A 的所有虚拟信息使 B 有办法检查她在步骤(10)茫然传送的有效性,迫使 A 在步骤(11)至(13)保持诚实。随着同步合同签名协议的进行,需要 A 信息对的左半部分和右半部分完成协议。

6.4 秘密的同步交换

前面两个协议可用来实现更一般的秘密同步交换。

A 知道秘密 a;B 知道秘密 b。如果 B 告诉 A 秘密 b,A 愿意告诉 B 秘密 a。如果 A 告诉 B 秘密 a,B 愿意告诉 A 秘密 b。校园中常会遇到这类事,但无法妥善进行,比如:

(1)A:"如果你首先告诉我,我告诉你。"

(2)B:"如果你首先告诉我,我告诉你。"

(3)A:"不,你先。"

(4)B:"噢,好吧。"(B 耳语)

(5)A:"嘿! 我不告诉你。"

(6)B:"那不公平!"

用密码术能保证公正执行,整个协议就不再重复了。

6.5 安全的选举

在正规选举中使用计算机化投票,要求能做到防止欺骗并保护投票人的秘密。理想的协议至少有五个特性:

1. 仅仅合法的投票者能投票;

2. 每人只能投一票;

3. 没有人决定投票人投谁的票;

4. 唱票之前没有人能改变其它任何人的票面;

5. 所有的投票者确信他们投的票已统计在最终的投票总表中。

另外,一些投票方案可能有下列需要:

6. 每个人知道谁投了票,谁没有投票。

在开始分析具有这些特性的复杂的投票协议之前,我们先看一些不具有此特性的简单协议。

简单投票协议#1

(1)所有的投票者用中心记票机构(CTF)的公钥加密他们的选票。

(2)所有的投票者把他们的选票发给 CTF。

(3)CTF 解密选票,记票并公布结果。

此协议在每点都有问题。CTF 无法知道投票者来自何地,因而不知道选票是否来自合格的投票者。无法知道合格的投票者是否多次投票。由于对你所选择的目标十分容易作无限次的投票,无人再会试图去修改他人的投票。

简单投票协议#2

(1)所有的投票者用他们的私钥签名他们的选票。

(2)所有的投票者用 CTF 的公钥加密他们的签名选票。

(3)所有的投票者将他们的选票发送给 CTF。

(4)CTF 将选票解密,检查签名,记票,并公布结果。

这个协议满足特性 1 和 2:仅仅合法的投票者能投票,每个人只能投一票。CTF 必须保持步骤(3)收到的投票记录。投票者用私钥签名投票,使 CTF 知道谁投票谁没有投票,不知道投谁的票,知道每一个投票者投了多少票。如果选票签名不正确或选票来自已投票者,记票机构将不于理睬。虽然在步骤(2)可截获选票,但由于数字签名,人们不能将它改变。

协议的问题是因为签名附在投票上,CTF 知道谁投谁的票。用 CTF 公共密钥加密投票防止其它人泄露并了解谁投谁的票,但这并不理想,因为 CTF 类似于一个选举监票人,他可看到整个选举的情况。

盲签名的投票

我们需要把投票者与选票分开,而仍可验证。用盲签名可完成这一任务。

(1)所有的投票者生成 10 组信息,每组对每一投票可能有一张有效的选票(例如,投票是"Yes"或"No"的方式,含有两张选票,一为"Yes"另一为"No")。每一个信息也含有随机生成的序列号,大到足以避免与其它投票者重复。

(2)所有投票者单独地盲签名所有信息,并把结果发送给 CTF。

(3)CTF 检查它的数据库,确信投票者没有重复投票。对组内,每一信息分别签字。然后回送到投票者并把投票者的名字存在数据库中。

(4)投票者打开盲签名,留下一组经 CTF 签字的选票(这些选票有签名但不加密,投票者容易地看见哪一张选票是"Yes"哪一张为"No")。

(5)投票者选择一张选票,并用 CTF 的公共密钥加密。

(6)投票者发送他们的选票到 CTF。

(7)CTF 解密选票,检查签名,检查数据库处理重复序列号,保留序列号,记票。公布选举的结果,以及每一序列号和与其相联系的选票。

对上述例子没有理由要使用全盲签名协议。CTF 不必检查十组信息中的九个以验证 A 在签名之前她的投票是合法的。A 的兴趣是确保正确生成她的选票,否则不能记票。

恶意的投票者不能欺骗这个系统。盲签名协议保证他只有一次投票机会。如果他试图发两张相同的选票,CTF 在步骤(2)观察到序号重复,抛弃第二张。如果他在步骤(2)多重签名,CTF 在步骤(3)发现。因为他不知道机构的公共的密钥,不能自行生成选票。同样的原因,他不能改变和截获其它人的投票。

恶意的 CTF 不能了解个人怎样投票。在投票计算之前,因为盲签名协议防止 CTF 观看投票的序号,CTF 不能把签名的盲投票同最后计算的投票连起来。公布序号表和相联系的选票使投票者确信他们的投票被正确地记票。

这一方案还存有问题。如果 CTF 记录下谁发送哪一张选票,那么它能了解谁投谁的票。然而,如果收到的选票锁在保险箱中,然后记票,就不能做了。此外,由于 CTF 可以不将选票与具体人相关联,它能签发大量有效的投票。

两个中央机构的投票方案

一种解决方法是在双方之间划分 CTF 的权力。做到没有一方具有足够的欺骗能力。

下列协议使用中心合法机构(CLA)证实投票者而另一个 CTF 做记票。

(1)所有的投票者发信息给 CLA 申请一个有效的序列。

(2)CLA 回送给投票者一个随机有效的序号 L。CLA 保留一张有效序号表。CLA 也保留

有效序号的收据表，防止重投。

(3)CLA 发有效序号表给 CTF。

(4)所有的投票者选择一个随机的、唯一的身份序号 I。用它和从 CLA 收到的有效序号以及选票 V 生成一条信息，并发给 CTF。

$$M=(L,I,V)$$

(5)CTF 从步骤(3)收到的序号表检查有效的序号。如果序号 L 存在，CTF 在它上面打上标记(防止任何人两次投票)。CTF 将该身份号加至某一后候人的选民表中，得票额总数加 1。

(6)所有的投票收齐后，CTF 公布结果，以及身份序号和他们投谁的票的列表。

这种系统的一个特点是所有投票者能看到身份序号的列表并找到他们自己。表明投票已记票。当然，协议中各方之间交往的所有信息被加密和签名，防止某些人模仿和窃听。

由于所有投票者将查看他们的身份序号，所以 CTF 不能修改投票。如果投票者未找到他们的身份序号，或在另一个不是他们投票的票中发现他们的身份序号，他们将立即知道投票中有违规行为。

由于 CLA 监视投票，所以 CTF 不能不把选票放入投票箱。CLA 知道多少投票者已被验证同时也知道他们的身份序号。两者中的任何一个被修改将立即被查出。不合法的投票者，通过猜测一个有效的身份序号可进行欺骗。然而如果可能的有效序号数远大于实际的有效序号数。其成功攻击的概率是很小的。当然，有效序号必须随机生成。

不管这些，CLA 在某些方面仍是一个可信任的仲裁者。它能识别不合法的投票者，识别合法投票者的多次投票。由于 CLA 公布了合法投票者(但没有身份序号)，风险可被减轻。如果投票人数少于记票数，则有出格之事。如果发现投票人数多于记票数，意味着可能一些合格者不愿投票(许多人登记投票，但没进行投票)。如 CLA 和 CTF 串通的话，协议是易受攻击的。他们在一起，对照数据库而发觉谁投谁的票。

单中心机构的投票

用更复杂的协议用来克服 CLA 和 CTF 之间沟通的危险。这种协议同前面的相同，但有两处修改。

- CLA 和 CTF 是一个组织，和
- 在步骤(2)中用 ANDOS 来分送匿名的有效序号。

因为匿名密钥分配协议防止 CTF 知道投票者得到哪一个有效序号，CTF 无法把有效序号和收到的选票联系起来。而我们仍信任 CTF 没有把有效序号分给不合法的投票者。

改进的单中心机构投票

它是一个非常好的协议，也使用 ANDOS，满足了投票协议的五个要求。它还不满足第六点要求，但有两个附加的特性：

(7)在给定的时间内，投票者能改变他们的主意(即收回他们的投票和再投票)。

(8)如果投票者发现他们的选票未记入，在不涉及他们投票秘密的情况下，可发现并纠正错误。

协议为：

(1)CTF 列表公布所有合法的投票者。

(2)在规定的期限内，打算投票的人向 CTF 汇报他们的意图。

(3)CTF 列表公布实际参加选举的投票者。

(4)使用 ANDOS 协议,所有投票者收到身份序号 i。

(5)所有投票者生成公钥/私钥对 e 和 d。如果 v 是选票,生成下列信息并发给 CTF:

$i, E_e(i, v)$

(6)CTF 通过公布下列信息回答收到选票:

$E_v(i, v)$ $E_e(I\ v)$

(7)所有的投票者将 I,d 发给 CTF:

i, d

(8)CTF 用 d 解密选票。在选举结束时,公布选举结果以及列表公布包含每一不同选票的所有 $E_e(i, v)$。

(9)如果投票者发现他们的选票没有正确地记票,他们发信息给 CTF 以示抗议:

$i, E_e(i, v), d$

(10)如果投票者想重改他们的投票 v′(有些选举是可能的),他们发送 CTF:

$i, E_e(i, v'), d$

步骤(1)至(3)为实际投票的准备阶段。目的是弄清并公布实际投票的总人数。尽管他们中的一些人可能不参加选举,但减少了 CTF 添加伪票的能力。

在步骤(4),两个投票者可能得到相同的身份序号。由于可能的身份序号远远大于实际的投票者,所以这种概率很小。如果两个投票者递交的投票具有相同的身份标记,CTF 生成一个新的身份序号 i′,选择一个用于投票并公布:

$i', E_e(i, v)$

票主认定它,用新的身份序号重复步骤(5),发送第二张选票。

步骤(6)给所有的投票者检查 CTF 正确地收到他们投票的能力。如果他们的选票没有记票,能在步骤(9)提出这种情况。假设在步骤(6)所有投票者的投票是正确的,步骤(9)发送的信息,构成选票未统计的证据。

协议主要的问题是 CTF 能分配步骤(2)已作响应但实际并未投票的人的选票。另一个问题是 ANDOS 协议的复杂性。作者建议把一群人划分为小的人群,如选举区。

6.6 安全的多方计算

安全的多方计算协议用于解决一组人在一起以特定的方式计算他们各自输入的函数值。组中每一个人都知道函数值,但不知道各人的输入。下面是几个例子:

协议#1

在彼此不知他人工资的情况下,如何计算一组人的平均工资?

(1)A 在她的工资中加一个秘密的随机数,用 B 的公钥加密并发给 B。

(2)B 用私钥解密 A 送来的数据并加上他的工资,然后用 C 的公钥加密并发给 C。

(3)C 用私钥解密 B 送来的数据并加上她的工资,然后用 D 的公钥加密并发给 D。

(4)D 用私钥解密 C 送来的数据并加上他的工资,然后用 A 的公钥加密并发给 A。

(5)A 用私钥解密 D 送来的数据。减去步骤(1)中给出的随机数,恢复成所有人工资的总和。

(6)A 用人数除以总和(在这种情况,人数为 4),公布结果。协议假设每个人都是诚实的。如果参与者说谎,平均工资是错误的。更严重的是 A 能谎报结果。为防欺骗,需要 A 使用任一比特递交方法给出随机数,但这样一来,B 将知道她的工资。

协议#2

A 和 B 同住旅馆，相谈谁的岁数大。他们不想把自己的年龄告诉对方，但又想知道谁年长。他们用耳语将各自的年龄告诉可信任的中立者(例如侍者)，让他比较后并把结果告诉 A 和 B。

上面的协议有两个问题。问题之一，一般侍者没有能力处理比这种大小比较复杂得多的计算。第二个问题，如果 A 和 B 真正关心他们信息的秘密性，他们将不得不防备中立者告知他人。

下面一个解决方案使用了公钥密码术。A 知道值 a，B 知道值 b，结合起来变为 f(a,b)的计算，因为使用的密码学算法 A 得不到有关 b 的信息，B 也得不到有关 a 的信息。A 和 B 相信计算的有效性。

当然，协议不能防止主动的欺骗者。无法阻止 A(或 B)谎报年龄。如果 B 是一个盲目地执行协议的计算机程序，A 重复执行协议就可探知 B 的年龄。A 执行协议，设她的年龄为 60 岁。知道她为年长之后，她用年龄 30 再执行协议。知道 B 年长后，她再用 45 岁执行协议，等等。A 可探知 B 的年龄准确到她所容的精度。

假设参与者不欺骗，这个协议容易扩展到多个参与者。依此诚实地执行协议，能找出他们年龄的大小顺序，而相互不知各人的年龄。

协议#3

议会在某些议案上作秘密投票，通常有如下规则：

例如所有 7 名议员以 yes 或 no 投票。另外，有 2 人有作“超级投票”的选择：S—yes 和 S—no。如果愿意的话不必去用“超级投票”，仅用常规投票，则以多数票决定此议案。有一个或二个等同的超级投票情况时，常规投票不作考虑。两个超级投票结果相反时，超过半数的常规投票起作用。我们需要一个协议安全地完成上述投票规则。

用两个例子说明投票的过程。假设有五张普通投票，从 N1 到 N5，和两张超级投票：S1 和 S2。议案#1 的投票结果为：

S1	S2	N1	N2	N3	N4	N5
S—yes	未投	no	no	no	yes	yes

在仅有 S1 投票的情况下，结果为 yes。

议案#2 的投票结果为：

S1	S2	N1	N2	N3	N4	N5
S—yes	S—no	no	no	no	yes	yes

两张超级投票恰恰相反，由大多数常规投票 no 决定议案。如果不需要对是否由超级投票或常规投票决定这件事保密的话，这是一个容易做到的安全投票协议。如果保密是重要的话，则需要更复杂的安全多方计算协议。

在实际生活中有这种投票情况。例如，联合国组织中的安理会国家，比其它国家有更大的权力。联合国提案的表决就比较复杂。

安全电路计算

A 的输入为 a，B 的输入为 b。他们想联合计算一些普通函数 f(a,b)，使 A 不了解 B 的输入，同时 B 也不了解 A 的输入。安全多方计算的通用问题称为安全电路计算。A 和 B 能生成一个任意布尔电路。这个电路接收 A 和 B 输入并产生一个输出。安全电路计算协议完成三件事：

1. A 送入她的输入值，不让 B 知道。

2. B 送入他的输入值，也不让 A 知道。

3. A 和 B 双方都能计算输出值，双方确信输出正确，没有一方作篡改。

6.7 数字现金

现金使人烦恼，它传布细菌又会失窃。信用卡和借贷卡大大地减少了现金在社会上的流通，但完全取消现金是不可能的。信用卡伴随着跟踪检查，钱的来往不能隐瞒，为此，人们需要交换金钱而无迹可寻的方法。

很幸运，有一个可信但不跟踪的协议。A 把数字“现金”转给参议员 B，但新闻记者 E 查不出 A 的身份而不能曝光。B 把电子现金存入到他银行的帐号上，尽管银行不知 A 是谁。但如果 A 试图将同一张数字现金使用两次，她将会被银行查出。同样，如果 B 企图把相同的一张数字现金二次存入，他也将被检测出但 A 保持匿名。

协议＃1

这是一个匿名现金汇票制备的实际方法，可被模拟成密码协议。

(1)A 以 1000 元准备了 100 份匿名现金汇票。

(2)A 把每一份和一张复写纸放入 100 个不同的信封中。她把他们交给银行。

(3)银行打开 99 个信封，确认每一份现金汇票为 1000 元。

(4)银行在保留未拆开的信封上签字。签名通过复写纸印到现金汇票上。银行把未拆开的信封返回给 A 并从她的帐号减去 1000 元。

(5)A 拆开信封，并将现金汇票付给一个商人。

(6)商人检查银行的签名，确信现金汇票是合法的。

(7)商人把现金汇票拿到银行。

(8)银行验证他的签名，把 1000 元存在商人的帐号上。

这个协议行得通。银行从不看签署的现金汇票，当商人把它带到银行时，银行不知道它是 A 的。由于有签名，银行确认它是有效的。基于分割选择协议，银行在签字时相信未打开信封的现金汇票为 1000 元。银行验证了其它 99 个信封，使 A 仅有 1% 的机会欺骗银行。当然，银行对诈骗行为进行高额罚款使 A 不值得在金额上进行诈骗活动。如果她冒险做了，将会遇到极大的不幸。

协议＃2

前面的协议防止 A 写入的现金汇票多于她的申报额，但不能防止 A 照相复制现金汇票并两次使用。对此，我们需要完成：

(1)A 为 100 个匿名现金汇票各准备 1000 元。在每一张现金汇票中引入一个不同的随机的、唯一的字符串，字符串足够长避免与他人使用的重复。

(2)A 把每张汇票、复写纸放入 100 个不同的信封中。她把所有的信封交给银行。

(3)银行打开 99 个信封，确认每一张现金汇票为 1000 元，所有随机的唯一的字符串是不同的。

(4)银行签剩余一个未拆开的信封。签名通过复写纸印到现金汇票上。银行把未拆开的信封返给 A 并从她的帐号中减去 1000 元。

(5)A 打开信封，并以现金汇票同商人做交易。

(6)商人检查银行的签名，确认现金汇票是合法的。

(7)商人把现金汇票拿到银行。

(8)银行验证他的签名,检查它的数据库确认具有唯一的字符串的现金汇票以前没存入银行。如果没有,银行把1000元存入商人的帐号。银行在数据库中记下此随机串。

(9)如果以前已存入,银行不接受现金汇票。

如果A或商人企图存入现金汇票的照相复制品,银行将查知。

协议#3

前面的协议防止了银行受诈骗,但不能识别诈骗者。银行不知道拿银行现金汇票的人是否企图诈骗商人,是否企图诈骗银行。协议改为:

(1)A为100个匿名现金汇票各准备1000元。在每一张现金汇票中包含一个不同的随机的、唯一的字符串,字符串足够长避免与他人使用的重复。

(2)A把每张汇票、复写纸放入100个不同的信封中。她把所有的交给银行。

(3)银行打开99个信封,确认每一张现金汇票为1000元,所有随机的唯一的字符串是不同的。

(4)银行签剩余的一个未拆开的信封。签名通过复写纸印到现金汇票上。银行把未拆开的信封返给A并从她的帐号中减去1000元。

(5)A打开信封,并以现金汇票同商人做交易。

(6)商人检查银行的签名,确认现金汇票是合法的。

(7)商人要求A在现金汇票上写上随机的身份字符串。

(8)A应允。

(9)商人把现金汇票拿到银行。

(10)银行验证他的签名,检查数据库确认该现金汇票以前有没有存入银行。如果没有,银行把1000元存入商人的帐号。银行在数据库中记下唯一的字符和身份字符串。

(11)如果唯一的字符串已在数据库中,银行拒绝接受现金汇票。然后,把现金汇票上的身份字符串和存在数据库中的字符串进行比较。如果相同,银行知道商人照相复制了现金汇票。如果不同,银行知道现金汇票的拥有者照相复制了汇票。

这个协议假设A只能一次性地在现金汇票上写上身份字符串,商人不能改变它。现金汇票可以有几个小的正方形,商人要求A填写X或O。现金汇票可由纸做,如作废即撕去就是。

因为商人同银行之间的交互行为发生在A花费以后,商人保留担心无效的现金汇票。协议的实施可能需要A在商人—银行交互时等待现金登记,与购买信用卡处理的方法非常相似。

A可用第二次复制的现金汇票与商人做交易进行欺骗,虽然在步骤(7)给出相同的身份字符串。但除非商人保有已收汇票的数据库,否则他将受骗。下一个协议克服了此问题。

协议#4

如果现金汇票持有者欺骗商人的企图暴露了,银行的责任是查明他是谁。做到这一点就要用密码术。

使用秘密分割技术把A的名字隐蔽在数字现金汇票中。

(1)A为100个匿名现金汇票各准备1000元。每一张汇票看似为:

数目:1000元

唯一的字符串:X

身份字符串: $I_1=(I_{1L},I_{1R})$

$$I_2=(I_{2L},I_{2R})$$

$$\vdots$$

$$I_{100}=(I_{100L},I_{100R})$$

每一张现金汇票包含一个不同的随机的唯一的字符串 X，其长度足以使两者相同的机会为 0。

在每一张现金汇票中，也有 100 对身份位串 I_n（每一张支票有不同的 100 对）。每一对如下产生：A 生成的含有她的名字、地址和银行想看的任何有关身份的信息的字符串。然后，使用秘密分裂协议，把字符串分成两部分。使用位递交协议，递交每一条字符串。

例如；I_{37}包含两部分：I_{37L}和 I_{37R}。每一对是一个位递交报文，它要求 A 来打开同时谁打开能立即验证。任一对的结合：I_{37L}和 I_{37R}，（但不能为 I_{37L}和 I_{38R}）揭示出 A 的身份。

(2)A 使用盲签名协议，签名所有的 100 张现金汇票。她把所有的交给银行。

(3)银行要求 A 对 99 张现金汇票揭开盲签名，确认所有的为合法的。银行检查钱数、唯一的字符串，要求 A 展示所有的身份字符串。

(4)如果银行满意地看到 A 没有进行诈骗，银行对剩余的一张盲现金汇票签字。并把盲签名现金汇票返给 A 从她的帐号中减去 1000 元。

(5)A 将现金汇票揭开盲签名，用它同商人做交易。

(6)商人检查银行的签名，确认现金汇票是合法的。

(7)商人要求 A 随意地展示现金汇票上每一个身份字符串左边一半或右边一半。实际上，商人给 A 一个随机的 100 位选择字符串，$b_1,b_2,\cdots,b_{100}$。A 取决于 b_i 是 0 或是 1，打开 I_i 的左边一半或右边一半。

(8)A 答应并完成上面要求。

(9)商人把现金汇票拿到银行。

(10)银行验证它的签名，检查数据库确认该现金汇票以前有没有存入银行。如果没有，银行把 1000 元存入商人的帐号。银行把唯一的字符串和所有身份信息记录在数据库中。

(11)如果唯一的字符串已在数据库中，银行拒绝接受现金汇票。然后，把现金汇票上的身份字符串和存在数据库中的字符串进行比较。如果相同，银行知道商人复制了现金汇票。如果不同，银行知道现金汇票的拥有者复制了汇票。因为第二个接收现金汇票的商人给出不同于第一个商人的 A 身份字符串，银行找出比特位，它使一个商人要 A 打开左边一半而另一个商人要 A 打开右边一半。将两个一半异或在一起，银行揭示了 A 的身份。

这是一个惊奇的协议，让我们从各个角度观察它。

A 能进行诈骗吗？她的数字现金汇票只不过是比特串，她能复制。第一次使用不会有问题；她将完成协议，每次平滑过渡。商人在步骤(7)给她 100 位随机选择字符串，步骤(8)A 打开 I_i 的左边或右边一半。在步骤(10)，银行将记录所有这些数据以及汇票的唯一的字符串。

当她试图第二次使用同样的数据现金汇票时，在步骤(7)商人（同一个商人或不同的商人）给她不同的随机的身份字符串。A 必须在步骤(8)应答；不做应答立即会被商人猜疑然而作了应答后，当商人在步骤(10)把现金汇票带到银行时，银行将注意到具有相同的唯一的字符串的现金汇票已存入银行。银行比较打开的一半身份字符串。由于两个随机选择相同的可能性为 2^{100}分之一；绝不会发生。银行发现一对身份串，一半是第一次另一半是第二次打开的。把两个一半进行异或，输出 A 的名字。银行知道谁试图两次使用现金汇票。

协议不需要 A 等待商人和银行通过他们协议的部分环节。如果 A 想进行诈骗,银行知道她的身份。A 无法阻止她的身份被揭露。她不能改变她唯一的字符串或任何身份字符串,否则银行签名将无效。商人在步骤(6)将立即发现。

A 试图偷偷地把无效的现金汇票骗过银行,汇票上没有揭示她名字的身份字符串;或用了其它人的名字的身份字符串。她在步骤(3)骗过银行的可能性为 1%。,但是一旦发现其罚款将使她破产,A 不会去冒这一风险。

商人能进行诈骗活动吗?他的机会更少。他不能存两次现金汇票;银行将发现选择字符串重复使用。他不能冒充丢失汇票的 A;只有 A 能打开一个身份字符串。

即使 A 和商人沟通也不能进行诈骗银行的活动。只要银行用唯一的字符串签署现金汇票,银行保证仅让现金汇票使用一次。

银行能做什么?有没有发现存入银行的现金汇票是 A 签名的方法?步骤(2)至(5)的盲签名协议保护了 A 的签名。即使银行保存每一笔交易的完整记录,银行也无法找到其间的联系。更妙的是,银行和商人在一起也无法认出谁是 A。A 漫步在商店中,完全匿名地进行购物。

其它数字现金协议

Tatsuaki Okamto 和 Kaluo Ohta 列出典型的数字现金系统的六个特性:

1. 独立性。数字现金的安全性不取决于任何物理位置。现金能通过计算机网络进行传送。

2. 安全性。数字现金不能拷贝和再使用。

3. 秘密性(不可跟踪)。保护用户的秘密;没有人能了解用户和他们的购货之间的关系。

4. 离线付款。当用户使用电子现金购物付款时,用户和商人之间的协议离线执行。即,商店不需连到主机上处理用户的付款。

5. 传输能力。数字现金能传给其它用户。

6. 划分能力。一笔给定数目的数字现金能划分为几笔小数目的数字现金(当然,每笔最终相加为一适当的数目)。

这里讨论的协议满足特性 1、2、3 和 4,但不满足 5 和 6。在线数字现金系统满足除特性 4 以外的所有特性。第一个离线数字现金系统满足特性 1、2、3 和 4,类似这里讨论的协议。T. Okamoto 和 K. Ohta 提出一个满足特性 1 至 5 的系统,他也提出一个满足特性 1 至 6 的系统,但购物需要的数据近似为 200MB。

同一作者推荐的数字现金方法满足特性 1 至 6,无需大量的数据。传送付款的总数据大约为 20KB,协议在几秒内完成。作者认为这是第一个理想的、不能跟踪的电子现金系统。

数字现金和完美的犯罪

数字现金也有阴暗的一面,有时人们要求太多的隐私。看看 A 实施完美的犯罪过程:

(1)A 绑架人质。

(2)A 准备了 10,000 张 1000 元的匿名现金汇票(她想要得到的钱款)。

(3)A 使用盲签名协议,盲签所有 10,000 张现金汇票。她把所有汇票送给当局,威胁除非满足如下条件,否则杀死人质:

(a)让银行在所有的 10,000 张现金汇票上签字。

(b)在报纸上公布结果。

(4)当局答应。

(5)A 买一张报纸,对现金汇票,揭开盲签名,开始花费它们。当局无法通过跟踪现金汇票跟踪她。

(6)A 释放人质。

注意：这种情况比任何随意的情况都糟，包括实际的代价券—例如现金。因为不需要实际的接触，警察没有机会接近绑架者。

实际的数字现金

一家荷兰公司 Digicash 在生产的产品中已实施了数字现金协议。感兴趣的人可按下述地址联系：

Digicash BV
Kruislaan 419
1098 VA Amsterdam
NETHERLANDS

6.8 匿名信息的传播

你同一群密码学者共进晚餐，会引起一场争吵。下面就是 David Chaum 引入的密码学者的晚餐问题：

三个密码学者坐在他们喜欢的三星级宾馆吃晚餐。侍者通知他们宾馆总管已安排好匿名付款。三个密码学者之一可能付晚餐的钱，或可能由国家安全局 NSA 付款。三个密码学者相互信任他们中的任一个会做匿名付款，但他们对 NSA 是否会付款感到没有把握。

名为 A、B 和 C 的密码学者能否决定他们中的某一位付款，同时对付款者的姓名保密？

Chaun 解决了这个问题：

每一个密码学者在他的菜单后面翻一个专用的硬币，位置的安排使得仅有他和他右边的密码学者两人能看到结果。每个密码学者要大声说他看到的两个硬币：他翻的一个和左邻翻的一个是否"相同"或"不同"。如果某密码学者为付款者，他报出的和他看到的正巧相反。说出在桌上报出的"不同"数为奇数，则表明由密码学者付款；"不同"数为偶数，则表明由 NSA 付款（假设晚餐仅付一次款）。如果由一个密码学者付款，其他两个人从桌面报出的信息不知谁付款。

为了看如何工作，假设 A 试图了解哪一个密码学者付款（假设不是她或 NSA 付款）。如果她看到两个不同的硬币，其它两个密码学者 B 和 C 中的任一个说："相同"或两者说"不同"（记住，密码学者说"不同"的数目为奇数表明他们中一个付款）。如果两者说"不同"，则付款者必定是投币结果与 A 所未见的投币结果相同者，如果两者说相同，则付款者必定是投币结果与 A 所未见的投币结果不同者。反之若 A 看见二个投币结果相同，不管 B 说"相同"，C 说"不同"或 B 说"不同"，C 说"相同"，只要 A 所未见的投币结果与她所能见的一样，则付款者必定是报"不同"的人，而 A 所未见的投币结果与她所能见的相反，则付款者必定是报"相同"的人。A 只有知道 B 和 C 投币结果才能知道谁是匿名付款者。

此协议能应用于多个密码学者中；他们坐成一圈，掷硬币。甚至两位也能完成协议；当然，他们知道谁付款，但旁观者只能说出是密码学者或 NSA 付款；不能说出哪一位密码学者付款。

此协议的应用超出了解决宾馆用餐的问题。这是无条件发送和无跟踪接收的一个例子。网络上一组用户使用这个协议发送匿名信息。

(1)用户自己构成一个逻辑环。

(2)他们之间有规则间隔的相邻用户掷硬币使用公正的掷币协议。他们确信所有信息是加

密的，无人了解他们掷币的结果。

(3)有规则间隔的每个用户说“相同”或“不同”。

如果用户想单独地广播信息，他们简单地把循环中的陈述由相应的二进制表达信息来控制。例如：如果信息为“1001”，则报告为与“与事实相反”“真实”，“真实”“与事实相反”。假设掷币的结果是“不同”、“相同”、“相同”、“相同”，他们说“相同”、“相同”、“相同”、“不同”。

如果用户发现协议的最终结果与他们试图发送的信息不匹配，他们知道也有人试图发送信息。他们停止发送信息，在再次发送之前，随机等待几圈。实际的参数必须依据网络上的信息流量计算出来，但概念应该是清楚的。

为了使事情更有趣，这些信息可用其中一个用户的公钥加密。当每一个人收到信息时(实际实施是加一些标准的开始信息和结束信息字符串)，只打算解密并阅读。没有人知道谁发送的。也没有知道谁读它。即使做流量分析也无助于跟踪和收集人们通信的方式。

第二部分　密码技术

第7章　密　　钥

7.1　钥　　长

对称密码系统的安全性决定于两个因素：算法强度和钥长。本节主要说明密钥长度在密码安全中的作用。

假设算法强度是完美的（尽管实际上很难做到，但易于用来说明钥长的作用）。所谓完美，即指除穷举搜索密钥（又称蛮力攻击）之外，没有更好的方法破译密码系统。如果钥长为 8 位，则密钥可能有 $2^8=256$ 个，所以要找到正确的密钥需做 256 次测试，测试期望值为 128。如果钥长为 56 位，则密钥可能有 2^{56}个。假设一台超级计算机每秒可测试 100 万个密钥，则找到正确的密钥需运行 2000 年。如果钥长为 64 位，则同一台计算机在 2^{64}的密钥空间中找到正确的密钥需运行约 600,000 年。如果钥长为 128 位，则需要 10^{25}年。宇宙年龄只有 10^{10}年；所以 10^{25}年是足够了。对于 2048 位长的密钥，一台每秒测试 10^{12}个密钥的并行计算机也得运行 10^{597}年才能找到正确的密钥，到那时宇宙也许早已不复存在了。

是否设计钥长为 8192 的密码系统就无懈可击了呢？请不要忘记它的强度还取决于另一个因素：算法必须坚固到没有比蛮力攻击更好的破译方法。这不是容易做到的，密码编制学是门精细的艺术，貌似完美的密码体制往往十分脆弱，坚固的密码体制只要有一点小小的改动就会变得脆弱。如果拙劣地同时使用两个坚固的密码体制，可能会相消而产生易于攻破的结果。对于密码编制人员的忠告是对一切新算法持怀疑态度，信赖一个算法最好是出于职业密码编制人员历经数年审查也未能攻破的经验，而不是由于算法设计者对其安全性的夸夸其谈。

由此引出另一要点：密码体制的安全性应在于密钥而不是算法细节。如果一个新密码体制的强度依赖于攻击者不知道算法内部操作细节，这将必败无疑。如果坚信对算法内部操作细节保密要比让学术团体来分析更能提高密码体制的安全性，也是大错特错的。认为没人能将源代码反汇编出来或将算法分析出来是极其天真的想法，诚然美国国家安全机构（NSA）对外保密他们的算法，但他们有世界一流的密码编制专家，另外，他们相互讨论他们的算法，以对等检查（peer review）的方式发掘他们工作中的弱点。

如果密码分析者得到了算法的所有细节，得到了完备的唯知明文攻击的密文，如果他们有了作明文攻击的足够数据，并选择了明文攻击，这时被攻击的密码系统还能保持安全，那么它具备了所需要的安全性。

在现实世界中的许多情况下，并非真正需要这里讨论的这种安全性。大多数敌手并没有政府所具有的知识和计算资源，甚至他对破译你的密码系统根本不感兴趣。

蛮力攻击的时间和费用估算

如果你认为对于一个算法最有效的攻击方法是蛮力攻击（极端的假设），那么密钥必须多长才使攻击不可实现呢？

蛮力攻击的速度取决于二个参数：要搜索的密钥数量和测试速度。大多数对称算法接受定

长位模式的密钥，DES 密钥为 56 位，密钥空间大小为 2^{56}（密钥其实是 64 位，其中 8 位作奇偶校验位）。本书所讨论的一些算法的钥长 64 位，有 2^{64}个可能的密钥，其它也有 128 位的密钥。

密钥的测试速度也是一个因素，但它不是主要因素。本分析中假设不同算法测试时间相同，实际上也许某个算法测试要比其它算法快二倍、三倍甚至十倍，但由于我们着重分析钥长，它使算法的攻破要比可行难上数百万倍，测试速度的微小差异就被忽略了。

密码学界中有关蛮力攻击效率的讨论，集中在 DES 算法上。1977 年 W. Diffie 和 M. Hellman 假设存在专用的 DES 攻击机，这种机器由 100 万个芯片构成，每个芯片能每秒测试 100 万个密钥，因此在 20 小时内可测试 2^{56}个密钥，如果用来攻击 64 位长密钥的算法，则在 214 天内可测试所有的 2^{64}个密钥。

蛮力攻击是为并行处理器特制的，每一个处理器可搜索密钥空间中的一个子集，处理器之间不必相互通讯，唯一的通讯机构是标识攻击成功的单一个报文。也不需要共享存储器。设计一台由 100 万个独立工作的并行处理器构成的机器，这样的机器是容易做到的。

每秒能测试 100 万个密钥的专用芯片也并非难以设计，现在的技术可以做到每秒测试 400 万个密钥的芯片：

“从过去 20 年的半导体制造历史来看，估计在五年之内这种芯片（每秒能执行 32,000,000 次加密操作）将问世。按照目前最先进的工艺技术水平，研制起码每秒能测试 4,000,000 个密钥的芯片是完全可行的。就现有的常备技术而言，每秒测试 200 万个密钥的芯片则是保证的。10 年之后应能设计出每秒测试 256,000,000 个密钥的芯片。”

把 4000 个每秒能测试 25600 万个密钥的芯片排成阵列，所有 56 位长的密钥可在 24 小时内测试完毕。800,000 个芯片则在同样时间内测试所有 64 位长的密钥。如果要保证数据在 10 年以上的安全性，这种钥长将远远不够。

表 7.1 给出在某个期限内对给定钥长做蛮力攻击所需的并行处理器的数目（假设了不同的处理器速度）。

表 7.1　在给定时间内对不同钥长做蛮力攻击所需的处理器数目

每秒执行的加密次数	在某时间内攻破 56 位长密钥所需的处理器数目			
（百万次）	1 年	1 个月	1 周	1 天
1	2,300	28,000	120,000	830,000
2	1,100	14,000	60,000	420,000
4	570	7,000	30,000	210,000
32	71	870	3,700	26,000
256	9	100	470	3,300
每秒执行的加密次数	**在某时间内攻破 64 位长密钥所需的处理器数目**			
（百万次）	1 年	1 个月	1 周	1 天
1	590,000	7,100,000	3.1×10^{7}	2.1×10^{8}
2	290,000	3,600,000	1.5×10^{7}	1.1×10^{8}
4	150,000	1,800,000	7,600,000	5.3×10^{7}
32	18,000	220,000	950,000	6,700,000
256	2,300	28,000	120,000	839,000
每秒执行的加密次数	**在某时间内攻破 128 位长密钥所需的处理器数目**			
（百万次）	1 年	1 个月	1 周	1 天
1	1.1×10^{25}	1.3×10^{26}	5.6×10^{26}	3.9×10^{27}
2	5.4×10^{24}	6.6×10^{25}	2.8×10^{26}	2.0×10^{27}
4	2.7×10^{24}	3.3×10^{25}	1.4×10^{26}	9.9×10^{26}
32	3.4×10^{23}	4.1×10^{24}	1.8×10^{25}	1.4×10^{26}
256	4.2×10^{22}	5.1×10^{23}	2.2×10^{24}	1.5×10^{25}

这些数字仅说明问题的一个方面，从概率上看，如果一台攻击机在一年之内可以找到密钥，那么将有8%的机会可以在一个月内找到密钥。如果密钥每个月更换一次，那么在本月内仍有8%的机会被攻破。如果这个密钥用于认证金融传送系统，问题可就严重了，即使敌手在密钥更换的十分钟之前获取了密钥，他就可以做巨额交易。

表7.2给出用蛮力攻击机在给定时间内找到某一密钥的概率。例如，一台在一个月内能找到密钥的机器有0.14%的可能性在一个小时内找到密钥。

表7.2　在给定时间内找到密钥的概率(假定做蛮力攻击)

蛮力攻击速度	1小时	1天	1周	1个月	1年
1天	0.0417	1.0000	1.0000	1.0000	1.0000
1周	0.0060	0.1429	1.0000	1.0000	1.0000
1个月	0.0014	0.0329	0.2308	1.0000	1.0000
1年	0.0001	0.0027	0.0192	0.0833	1.0000

经常性地更换密钥可以限制敌手已经获得某个密钥的情况下造成的危害，但在许多应用场合下这仍然是不可接受的风险。

支配蛮力攻击的另一个因素是费用问题。表7.3给出了各年研制蛮力攻击机、钥长和找到密钥所需时间的费用估计。1990年的估计值是采用每秒执行100万次和200万次加密的芯片的机器的平均值，假设每秒执行400万次加密的芯片过于昂贵而未采用。1995年的估计值是采用每秒执行200万次和400万次加密的芯片的机器的平均值，2000年的估计值是采用每秒执行400万次和3200万次加密的芯片的机器的平均值。

表7.3　蛮力攻击机的费用(单位：百万美元)

攻破56位算法的机器				
年份	1年	1个月	1周	1天
1990	0.13	1.6	6.7	47
1995	0.064	0.78	3.3	47
2000	0.016	0.190	0.83	5.8
2005	0.002	0.024	0.1	0.73
攻破64位算法的机器				
年份	1年	1个月	1周	1天
1990	33	400	1700	12000
1995	16	200	850	6000
2000	4.1	50	210	1500
2005	0.51	6.2	27	190
攻破128位算法的机器				
年份	1年	1个月	1周	1天
1990	6.0×10^{20}	7.3×10^{21}	3.2×10^{22}	2.2×10^{23}
1995	3.0×10^{20}	3.7×10^{21}	1.6×10^{22}	1.1×10^{23}
2000	7.5×10^{19}	9.2×10^{20}	3.9×10^{21}	2.8×10^{22}
2005	9.4×10^{18}	1.2×10^{20}	4.9×10^{20}	3.4×10^{21}

攻破56位和64位密钥的费用已在许多工业国家的军事预算之中。但攻破128位的密钥的费用仍然是远不能接受。

如果敌手要攻破一个密钥，他所做的就是投入必需的费用。所以估计一个密钥的最小"价值"似乎得格外谨慎：攻破一个密钥在产生"经济效益"之前需要投入多少？给出一个极端的例子：假设价值1.39元的明文用某算法加了密，投资400万来破译似乎不会有什么效益。另一方

面，如果明文价值10000万，则他也许“值得”夜以继日地独自破译。

表7.4给出攻击者在各个时期内要找到密钥的费用。如果在过去5年中，找到的每个密钥的价值与所支付的费用相当，攻击者基本持平。如果每个密钥的价值都超过支出，他就盈利了。这里假设攻击机昼夜不停地运行：每天24小时，每年365天。这里没有假设找到的密钥的价值都是最小的，而是平均值，只要有一个高价值的密钥就可使5年的劳作获益颇丰。

表7.4 在各个时期内找到密钥的费用(单位：百万美元)

攻破56位密钥的费用				
年份	1年	1个月	1周	1天
1990	0.047	0.046	0.045	0.045
1995	0.024	0.023	0.022	0.022
2000	0.0059	0.0057	0.0056	0.0056
2005	0.00074	0.00071	0.0007	0.0007
攻破64位密钥的费用				
年份	1年	1个月	1周	1天
1990	12	12	12	11
1995	6.1	6	5.8	5.7
2000	1.5	1.5	1.4	1.4
2005	0.19	0.18	0.18	0.19
攻破128位密钥的费用				
年份	1年	1个月	1周	1天
1990	2.2×10^{20}	2.2×10^{20}	2.1×10^{20}	2.1×10^{20}
1995	1.1×10^{20}	1.1×10^{20}	1.1×10^{20}	1.1×10^{20}
2000	2.8×10^{19}	2.7×10^{19}	2.7×10^{19}	2.6×10^{19}
2005	3.5×10^{18}	3.4×10^{18}	3.3×10^{18}	3.3×10^{18}

软件破译者

没有专用硬件和巨大的并行机，蛮力攻击是相当困难的。软件的速度实在太慢，以至于即使对56位的密钥，经济的蛮力攻击也不可行。Motorola 68040微处理器每秒大约可执行23,200次DES加密运算，这速度在过去的10年中已经提高了16倍，按照这个比例，微处理器在5年之内可以每秒执行64,000次DES加密，10年之内可以每秒执行256,000次，这仍然要比当前最好的硬件实现慢数千倍。

基于软件作蛮力攻击的真正威胁，不在于它一定能成功，而是因为它是“免费”的。用一台空闲的微机测试密钥空间并不需要投资，如果碰巧找到了正确的密钥则是最好不过的，如果没有破译也并不会损失什么。用整个微机网络去测试也不需要投资。最近一项实验在一天内用40台工作站的所有空闲时间测试2^{34}的DES密钥，按照这个速度，搜索整个密钥空间需要400万天，但如果足够多的攻击者参与了，也许会有人庆幸破译成功。

软件威胁的关键纯粹是运气。假如某大学计算机网络有512台工作站联在一起，这在校园网中也只算是中等规模的网络，它们甚至可以分布在世界各地，通过电子邮件协同工作。假如每一工作站每秒执行15,000次加密，扣除测试和更换密钥的开销，每秒执行8,192次测试，穷举56位的密钥空间需545年(假设网络每天24小时投入测试)，这将在一天内有1/200,000的机会攻破一个密钥，经过一个星期，破译成功的机会增加到1/66,000。如果硬件速度更快，或者投入更多的机器，成功的机会将会更多。

攻击64位的密钥比攻击56位的密钥难256倍，与破译40位的密钥相比，前景将十分暗淡。尽管由400台每秒执行32,000次加密的计算机组成的网络能够在一天内对40位的密钥

完成蛮力攻击(在1992年,美国批准RC2和RC4算法以40位密钥出口)。

攻击128位的密钥显得十分荒唐,估计1996年全球从Cray巨型机到笔记本共有200万台计算机,如果所有的计算机一起参与蛮力攻击,每台计算机每秒执行100万次加密,运行时间也须大于宇宙年龄的100万倍才能破译。

病毒

让数百万台计算机一起参与蛮力攻击的主要困难也许是征得计算机的主人同意,虽可礼貌地一一请求,但极为费时,更糟糕的是被拒绝。或许在夜晚像黑客一样闯入他们的计算机,但这是违法的,会遭逮捕。有人用计算机病毒来有效地传播攻击程序。

这是特别狡猾的主意。攻击者编写并释放了一个计算机病毒,这个病毒并不格式化硬盘或删除文件,而是让空闲着的计算机运行蛮力攻击密码分析程序。调查表明微机在70%到90%的时间是空闲的,于是病毒轻而易举地占用了去执行它的任务。如果病毒是良性的,它执行任务时也许并不会引起注意。

最后,某台计算机碰巧找到了正确的密钥,这时有两种处理方式。第一,病毒繁衍出另一个病毒,它删除攻击病毒的所有副本并且带有正确密钥的信息,新病毒在计算机空间内繁殖,直到到达编写源病毒的计算机。

第二种方式是在屏幕上显示如下"悬赏"信息:

There is a serious bug in this computer.
Please call xxxxxxxx and read the following
64-bit number to the operator:
xxxx xxxx xxxx xxxx
There is a $100 reward for the first person
to report this bug.

这种攻击方法效率如何?假设典型的感染上病毒的计算机每秒能测试1000个密钥(这远远低于计算机的最大性能,因为假设计算机偶尔也得执行一些其它的任务),另外假设病毒感染了14万台计算机,则在83天内即可攻破56位的密钥,在1958年内攻破64位的密钥。当然加快计算机速度和提高病毒感染率都将加速攻击。

生物工程技术

如果生物芯片问世,无疑是分布式蛮力攻击密码分析的理想工具。假设有密钥恐龙(DES-osaur)一类的动物,它由能够测试密钥的生物细胞构成,明文/密文对通过光信道传播到各个细胞(细胞是透明的),结果通过循环系统传递到密钥恐龙的语言器官。

一般恐龙大约有10^{14}个细胞(不含细菌),如果每个细胞每秒能执行100万次加密运算,破译56位的密钥需要7/10,000秒,破译64位的密钥需要2/10秒,而破译128位的密钥需要10^{11}年。

另一种生物方法是用基因工程生产能作蛮力攻击算法的密码分析水藻,这种有机体使构造具有更多处理器的分布式机器成为可能,因为它们可以覆盖大面积区域。明文/密文对可以通过卫星播放,如果某个有机体发现结果,它可使附近的细胞改变颜色以把结果回送给卫星。

假设水藻细胞直径为10微米,则每立方米内有10^{15}个细胞。将水藻倒入大海覆盖518平方公里的水域并深度为1米,将有10^{23}个细胞漂浮在海面。如果每个细胞能每秒测试100万个密钥,100多年即可穷举128位的密钥空间。

上述只是理论分析,不去问及技术实现细节。

密钥应有多长?

答案是与应用场合有关。数据需要确保安全多久?数据价值如何?甚至也可以此确定安全性需求,例如:根据必须保证攻击者耗费 10000 万美元,并且每年递增 30%投入才能以成功率小于 $1/2^{32}$破译系统的安全性需求,可以选用某一钥长。

表 7.6 给出对不同信息的安全性需求估计。

表 7.6 不同信息的安全性需求

信息类型	生存期	最小钥长(比特)
军事战术信息	分/小时	56
产品宣告、合并、利息率	天/周	56-64
商业机密(如可口可乐配方)	10 年	64
氢弹机密	>40 年	128
间谍身份	>50 年	128
个人隐私	>50 年	128
外交窘态	>65 年	>128
美国联邦统计数据	100 年	>128

未来计算机的性能是难以估计的。这里给出一个合理的粗略估计方法:计算机性能价格比以每 5 年 10 倍的因子上升。这样 50 年后最快的计算机速度将是今天的 10^{10}倍,并且这个数字仅就通用计算机而言,谁知道 50 年后是否会研制出破译密码系统的专用计算机呢?

假设一个密码算法要使用 30 年,据此你可以想象它该具有何等的安全钥长。必须考虑到现在设计的算法也许直到 2000 年还未普遍使用,但可能在 2030 年用来加密信息并直到 2080 年还保证安全。

7.2 密钥管理

A 和 B 有一个安全的通讯系统,他们玩心理扑克、同时签署合同、甚至一起放心地评价电路。他们的协议是绝对安全的,他们的密码算法是一流的,但是他们的密钥是从 E 处买来的,E 的口号是"你可以信赖我们,安全性是我们的金字招牌"。

E 无需破译算法,不必依赖于协议中的细小缺陷,她轻而易举地就可得到 A 和 B 的信息交换资料。

现实世界中,密钥管理是最困难的安全性问题。设计安全的密码算法和协议并非易事,但保证密钥安全更为困难。密码分析者经常通过密钥管理来攻击对称和公开密钥的密码体制,如果从密钥管理手段着手可以轻而易举地得到密钥,又何必费尽心机地去攻击密码算法?至少,A 和 B 应该把密钥和数据的安全程度同等看待。如果用户习惯于以姓名作密钥或把密钥写在小纸片上夹在皮夹子中,世上最安全的算法也无济于事。

7.2.1 密钥生成

算法的安全性在于密钥。如果密钥由脆弱的密码程序生成,则整个系统都比较脆弱。当 E 能够分析密钥生成算法时就无需分析密码算法。

例如,Macintosh 机的 DiskLock 程序(2.1 版)声称具有 DES 加密的安全性,它用 DES 加密文件。虽然它的 DES 算法实现是正确的,然而,DiskLock 把密钥和加密后的文件存储在一起,如果能够在加密文件中找到密钥就可以对其解密,这和 DES 加密算法无关,只是实现方式实在太不安全了。

缩小的密钥空间

DES 的密钥为 56 位，适当地处理，任何 56 位的字符串都可用作密钥，密钥空间为 2^{56} (10^{16})个。如果 DES 软件加密程序只允许密钥由小写字母和数字组成，密钥空间将缩小到 10^{12} 个。如果密钥生成程序比较脆弱，则 DES 将更易于破译。

表 7.7 给出对密钥不同限制情况下密钥空间的大小。表 7.8 给出假设每秒测试 100 万个密钥情况下，穷举搜索各个密钥空间所需的时间。请注意密钥长短次序的区别说明穷举搜索 8 字节密钥空间与 4，5，6，7 字节密钥空间所需的时间差别并不大。

表 7.7　不同密钥空间的大小

	4 字节	5 字节	6 字节	7 字节	8 字节
小写字母(26 个)	4.6×10^{5}	1.2×10^{7}	3.1×10^{8}	8.0×10^{9}	2.1×10^{11}
小写字母和数字(36 个)	1.7×10^{6}	6.0×10^{7}	2.2×10^{9}	7.8×10^{10}	2.8×10^{12}
字母和数字(62 个)	1.5×10^{7}	9.2×10^{8}	5.7×10^{10}	3.5×10^{12}	2.2×10^{14}
可打印字符(95 个)	8.1×10^{7}	7.7×10^{9}	7.4×10^{11}	7.0×10^{13}	6.6×10^{15}
ASCII 字符(128 个)	2.7×10^{8}	3.4×10^{10}	4.4×10^{12}	5.6×10^{14}	7.2×10^{16}
8 位 ASCII 字符(256 个)	4.3×10^{9}	1.1×10^{12}	2.8×10^{14}	7.2×10^{16}	1.8×10^{19}

表 7.8　穷举搜索各个密钥空间所需的时间

	4 字节	5 字节	6 字节	7 字节	8 字节
小写字母(26 个)	0.5 秒	12 秒	5 分	2.2 小时	2.4 天
小写字母和数字(36 个)	1.7 秒	1 分	36 分	22 小时	33 天
字母和数字(62 个)	15 秒	15 分	16 小时	41 天	6.9 年
可打印字符(95 个)	1.4 分	2.1 小时	8.5 天	2.2 年	210 年
ASCII 字符(128 个)	4.5 分	9.5 小时	51 天	18 年	2,300 年
8 位 ASCII 字符(256 个)	1.2 小时	13 天	8.9 年	2,300 年	580,000 年

这里可以采用各种专用蛮力攻击硬件和并行技术，无论是对于一台机器还是多台机器并行处理，如果每秒测试 100 万个密钥，破译 8 字节以下小写字母密钥和小写字母与数字构成的密钥、7 字节以下字母数字密钥、6 字节以下可打印字母密钥和 ASCII 字符密钥以及 5 字节以下 8 位的 ASCII 字符密钥是可能的。

另外，计算机总归是越来越快且越来越便宜，就是说，不要指望密钥能够承受蛮力攻击十年之久。

弱钥选择

人们选择的密钥往往是个脆弱者。例如与“＊9(hH\A-”相比，人们更可能选择“Barney”，这倒不是安全性实践不够的缘故，而主要是“Barney”比“＊9(hH\A-”好记得多。精明的蛮力攻击者并不会依次搜索密钥空间，而是首先测试最容易的密钥：英语单词、姓名、英语单词的细小扩展等等，这称为字典攻击(dictionary attack)，因为攻击者使用普通密钥的字典。D. Klein 在普通计算机上根据下列策略可以攻破 25%的口令：

1. 试探用户姓名、字首、帐号名和其他相关人员的信息。总的说来，基于这个思想最多需测试 130 个口令。例如用户 Daniel Klein 的帐号为 Klone，需测试的部分口令为：Klone，Klone0，Klone1，Klone123，dvk，dvkdvk，dklein，DKlein，leinad，nielk，dvklein，DvkkvD，DANIEL，KLEIN，KleinD 等等。

2. 试探来自不同数据库的单词。包括男女姓名表（共 16,000 个）、地名（包括排列，如“spain”，“spaish”和“spainard”等）、名人姓名、卡通人物、电影和科幻小说的标题、人物和地点、体育（包括队名、绰号和专用术语）、数字（数字“2001”和英语单词“twelve”）、数字和字母串（“a”，“aa”，“aaa”等）、汉语音节（汉语音节的拉丁化表示，这是在英文键盘上书写汉字的国际标准）、圣经、生物术语、大众俗语（如“deadhead”等）、键盘排列（如“qwerty”，“asdf”和“zxcvbn”）、缩写（如“roygbiv”表示彩虹的色彩）、机器名（取自/etc/hosts）、行星名以及各种技术报告中的单词表。所有这些，每个用户大约需测试 60,000 个单词。

3. 试探第 2 步中单词的某些变换。包括使首字符大写或用控制字符、使整个单词大写、颠倒单词（带/不带前述大写化）、把字母“o”改为“0”（单词“scholar”改为“sch0lar”），类似地把字母“l”改为“1”、字母“z”改为“2”、字母“s”改为“5”。另外还有把单词复数化，如“dress”改为“dresses”、“house”改为“houses”，也可以不管其复数形式是否正确，如把“datum”改为“datums”（不是“data”）。类似地也可以在单词后面添加“-ed”、“-er”、“-ing”等后缀，如把“phase”改为“phased”、“phaser”和“phasing”。这部分在测试空间中增加约 1,000,000 个单词。

4. 试探第 2 步中单词的各种大小写化。包括使单字符大写化（如“michael”改为“mIchael”、“miChael”、“micHael”、“michAel”等），双字符大写化（如“MIchael”、“MiChael”、“MicHael”、…、“mIChael”、“mIcHael”等），三字符大写化等等。单字符大写化在测试空间中增加约 400,000 个单词，双字符大写化增加约 1,500,000 个单词，三字符大写化增加至少 3,000,000 个单词。由于计算机性能的限制，一般不考虑 4,5 和 6 个字符大写化变换。

5. 试探用户母语单词。例如对中国用户测试其中文姓名（汉语音节的拼音拉丁化），由于无法确定这种单词是否有意义，只能对其穷举搜索。汉语音节有 298 个，故有 158,404 个双音节单词，大约 16,000,000 个三音节单词。根据英语中可发音但无意义的单词构造原则可简化这种攻击模式。

6. 试探单词对。这种穷举搜索的规模是不稳定的，为简化搜索，3 和 4 字符长的单词只使用/usr/dict/words 中的单词，即使如此，单词对还有 $O(10^7)$，而这里只测试了 10%。

字典攻击作用于密钥文件比作用于单个密钥更有效。某个用户也许会聪明地选择好的密钥，如果成千用户人人选择他们自己的密钥作为计算机系统的口令，至少有一个用户选择了攻击者字典中的密钥的可能性是很大的。

生成好密钥

好密钥是随机位串。如果钥长为 64 位，每一个 64 位密钥都很可能类似的。密钥位可从可靠的随机源也可从安全的伪随机位发生器生成。如果不能采用这种自动方法，也可以用投币或掷骰子的方法。

有些密码算法中存在弱密钥：比其它密钥的安全性差的密钥。如果发现一个弱密钥最好测试一下所有弱钥并重新生成一个密钥。DES 只有 16 个弱钥，正好生成这些弱钥的可能性是很小的。已经证明潜在攻击者并不知道使用了弱钥，所以偶尔用用也无妨。然而，测试这些少量弱钥太容易了，人们往往可以轻率地不去测试。

测试公钥密码体制的密钥要困难得多，因为密钥带有特定的数学模型（如素数、平方余数等）。从密钥管理角度来看，须注意的是随机数发生器的种子也必须是随机的。

生成随机密钥也并非总是可能的。有时用户得记住密钥（看看经多长时间才能记住 25E856F2E856F2E8BAC820），如果必须生成一个易于记忆的密钥，也必须把它搞的隐晦。理想的情况是易于记忆但又难以猜测。以下是几点建议：

· 用标点符号隔开单词对。如“turtle * moose”和“zorch！ splat”。

· 取一句短语各单词的首字符。如由“Mein Luftkissenfahrzeug ist voller Aale！”生成密钥“MLivA！”。

更好的解决方法称为“密钥粉碎(key crunching)”技术：把易于记忆文本串转换成随机密钥的技术。用单向哈希函数把任意长度的文本串转换为伪随机位串。

例如易于记忆的文本串为：

My name is Ozymandias, King of Kings. Look on my works,ye mighty,and despair.

可以粉碎成 64 位密钥：

EBC1 4398 5AE9 0A9B

文本串通常指的就是通行短语(pass phase)。只要短语足够长，结果密钥就显得随机。对于 64 位的密钥，10 到 15 个单词的通行短语应该是足够的。大致上每位密钥需要 1 个字符：64 位密钥需 64 个字符。这里仅假设对小写字母和空格而言，而没有大写字母和标点符号。要增加密钥的随机程度，在通行短语中可采用整个可打印 ASCII 字符集(大小写字母、数字、标点和空格)。

这种技术也可用于生成公钥密码体制中的私钥：文本串粉碎成随机种子然后把种子送入生成公钥/私钥对的确定系统。

X9.17 密钥生成

ANSI X9.17 标准说明了一种密钥生成方法。它不生成易于记忆的密钥，而是更适合于生成一个系统内的会话密钥。它生成密钥的密码算法是 DES，但也可是其它算法。

设 $E_k(X)$ 为 X 用密钥 k 的 DES 加密操作，保留密钥 k 用于秘密密钥生成。V_0 是秘密的 64 位种子，T 为时戳，要生成随机密钥 R_i，计算：

$$R_i = E_k(E_k(T_i) \text{ XOR } V_i)$$

要生成 V_{i+1}，计算：

$$V_{i+1} = E_k(E_k(T_i) \text{ XOR } R_i)$$

简单地把每个第 8 位当成奇偶校验位即把 R_i 转换成 DES 密钥，也可直接当成 64 位密钥使用。如果需要 128 位的密钥，只需生成一对密钥并把它们连接在一起。

7.2.2 密钥传送

A 和 B 欲用常规的密码算法安全通讯，他们需要同一密钥。A 用随机密钥产生器生成一个密钥，于是她得把密钥安全地交给 B。如果 A 和 B 能在某处(后院或密室)会面，她可将密钥副本交给 B。否则他们就面临一个绝对保密地传送密钥的问题。

A 可将密钥通过他们欲加密的通讯信道传送给 B，但这是愚蠢的做法：如果信道保证做加密操作，赤裸裸地传送加密密钥将保证窃听同一信道的所有人都能对所有 A 和 B 的通讯解密。

公钥加密系统较好地解决了这个问题，但这技术也并非一直可用，一些系统交替使用已知安全的信道。A 可把密钥和可靠的信息一起传送给 B，她可通过签发邮件或特快专递，或者她和 B 之间建立另一条但愿没人窃听的通讯信道。

更好的解决方法是把密钥拆分成几部分，每一部分通过不同的信道传送，例如可以一部分通过电话传送，一部分用信件传送，一部分用快件传送，一部分由信鸽传送等等，只要攻击者不把密钥的各个部分收集完整，他就无法确定密钥。这种方法有很多种工作方式。

然而，如果每天更换密钥，这就显得很不方便。对于 A 而言，一种解决方法是每个月定时地传送给 B 一个月的密钥表，这密钥表是很有价值的，A 和 B 都必须在收到后妥善保管，一旦泄露，全月的通讯都处于危险之中。

有的系统解决这个问题的方法是生成一个主钥(master key)，或称为密钥加密的密钥(key-encryption key)。A 无论是通过会晤或上述讨论的密钥拆分技术安全地把主钥传送给 B。一旦 A 和 B 都有了主钥，A 每天可通过同一个通讯信道把新密钥传送给 B，A 把每天的新密钥用主钥加密。由于以主钥加密的通讯量比较小，故无需经常更换主钥。然而，由于主钥泄露将导致泄露以主钥加密的密钥，进一步导致以密钥加密的所有信息，所以必须妥善保管主钥。

大网络上密钥分发

在小网络上一对用户可很好地共享主钥，但随着网络扩大就会显得不方便。由于每对用户必须交换密钥，n 个用户的网络上密钥交换量为 n(n－1)/2。

6 用户网络上密钥交换量为 15，1000 用户的网络上密钥交换量达到近 500，000。在这种情况下，建立中央密钥服务器将使操作有效得多。

7.2.3 密钥验证

当 B 收到密钥后，他如何知道密钥来自于 A 而不是冒充 A 的人？如果 A 和 B 面对面传送，这不成问题。如果 A 通过可靠的信使传送密钥，B 必须信赖信使。如果密钥以主钥加密，B 必须相信只有 A 拥有主钥。如果 A 用数字签名协议签署密钥，B 必须信赖公钥数据库。如果密钥分配中心(KDC)签署 A 的公钥，B 必须信赖他的 KDC 公钥副本没被篡改。

最后，如果有人控制了 B 所在的整个网络，他就可使 B 相信一切。攻击者可以发送加了密和签了名的信息，并且声称来自于 A。当 B 想访问公钥数据库来验证 A 的签名时，攻击者可以替换他自己的私钥。攻击者可以建立虚假的 KDC 并替换成真的 KDC 的公钥。B 对此一无所知。

有人以此为论据提出公钥密码系统无实用价值。因为 A 和 B 确信他们的密钥未被篡改的唯一方法是面对面传送，公钥密码系统并不能提高安全性。

这个观点是很天真的。尽管理论上是正确的，但实际上要复杂得多。采用数字签名和可靠的 KDC 的公钥密码系统使更换某人密钥非常复杂。B 不能绝对确信攻击者没有控制整个网络，但应相信攻击者要做到这样必须要访问到非同寻常的资源。

B 可以通过电话验证 A 的密钥。这是公钥：他可以通过电话叙述密钥。如果密钥过长，A 和 B 可以使用单向哈希函数验证密钥，这有很多方法。

有时严格地验证公钥究竟属于谁并不重要，而必须验证的是它与以前是否同属于一人。例如如果某人给银行发送签了名的提款信息，银行并不必关心是谁提款，而是关心他是否是储户。

错误检测

有时密钥在传送过程中会出错。出错的密钥意味着大量的密文无法解密，所以必须设法把这问题减少到最小程度。所有密钥传送时应带有错误检测和校正位，至少应附上检验和。这种方法可以容易地检测出传送中的错误，如果需要可以重发。

最为广泛采用的方法之一是连同密钥加密一段常量(全 0 或全 1)，再连同密钥发送 2 或 4 个密文字节。接收方执行类似的操作。如果加密常量正确，则密钥也正确地传送了。这种方法错误未检出率在 $1/2^{16}$ 到 $1/2^{24}$ 之间。

7.2.4 密钥使用

软件加密是不安全的。微机单任务时代已经过去，现在有 Macintosh 系统 7、WindowsNT 和 UNIX 等多任务系统。当操作系统挂起加密程序的进程，把所有内容写到磁盘再执行紧急任务时并不会提示用户。当操作系统最后回到加密进程时，无论当时加密什么内容，所有都似乎一切正常。没人会发觉操作系统把加密程序连同密钥一起写到磁盘，密钥将一直赤裸裸地存储在磁盘上，直到计算机对该存储区域覆盖，这当中可能是 1 分钟或 1 个月，甚至永远不会覆盖，当攻击者用搜索工具搜索硬盘时密钥还是原地不变。

在先到先服务的多任务环境下，可以把加密操作的优先级设置成不会被中断，这也减轻磁盘的负担。虽然如此，整个操作还是处于冒险之中。

硬件实现要容易一些。许多加密设备都设计成一旦遭到篡改即把密钥删除，例如 IBM PS/2 加密卡有一个包含 DES 芯片、电池和存储器的环氧单元。当然，用户得信任制造商正确地实现了功能。软件实现也应试着这么做，但完美者微乎其微。

7.2.5 密钥存储

最容易的密钥存储问题是用户对储存文件加密。由于只有一人参与此事，只有他对此密钥负责。一些系统采用简易方法：密钥存储在用户头脑中而永远不在系统中出现，用户负责记忆密钥并每当加密或解密文件时输入密钥。

这类系统的一个例子是 IPS。用户输入密钥时可以直接输入 64 位密钥，也可以输入字符串，系统用密钥粉碎技术从字符串生成 64 位密钥。

另一种方法是把密钥存储在 ROM 和磁卡中，用户可以通过把 ROM 或卡插入一种连接在加密盒或计算机终端的特殊阅读器而输入密钥。

如果把密钥拆分成二半，一半存储在终端上，另一半存储在 ROM 中，则这种技术会更安全。这样，即使丢失了 ROM 也不会泄露密钥，只需重新更换即可，终端密钥丢失了也一样。所以 ROM 中的密钥或系统中的密钥泄露都不会泄露整个密钥，攻击者必须取得两部分的内容才能重构原密钥。

难以记忆的密钥可以用类似于主钥的方法以加密的形式存储。例如，RSA 私钥可以用 DES 密钥加密并存储在磁盘上，要恢复 RSA 密钥，用户得把 DES 密钥输入解密程序。

如果密钥是确定生成的(用密码安全的伪随机数产生器)，也许每当需要时从易于记忆的口令重新生成密钥是比较容易的。

7.2.6 密钥备份

假设 A 是某单位的财政主管，像其他称职的办公人员一样，她遵循公司的安全制度把所有数据都加了密。不幸的是，她忽视了交通规则而被卡车撞了。公司经理 B 怎么办呢？

要是 A 未把密钥副本留给了 B，他可就麻烦了。没有密钥，文件就无法解密。除非 A 是个低能者并使用蹩脚的加密软件，否则她的文件就变成了“死”文件。

B 可以有许多避免这种情况发生的措施。最简单的方法就是要求所有雇员写下他们的密钥并交给公司安全人员，由安全人员把密钥锁在一个安全的地方(或者用主钥加密)。现在 A 遇了车祸，B 可向安全人员索要她的密钥。B 要保证这种行为是保险的，否则如果安全人员不在时，B 还是只能望“文件”兴叹。

其中的问题是 B 得信任他的安全人员不会误用大家的密钥，同时，每个雇员都得信任安

全人员不会误用他们的密钥。一个好得多的方法是使用秘密共享协议。

当 A 生成密钥时，她可把密钥拆分成许多片，她按照公司的安全制度把密钥的各片以加密的形式传送给不同的公司人员。没有一片可单独作为密钥，但有人可以把各片汇集起来重组成密钥。现在，A 的任务是防止恶意人士，而 B 预防 A 有麻烦时丢失 A 的数据，或者 A 可以把各片密钥以每个雇员的各自公钥加密后存储在自己的硬盘上，这样，无需谁介入密钥管理。

7.2.7 密钥泄露

只有当密钥（公钥系统中的私钥）安全时本书中的协议、技术和算法才是安全的。如果密钥丢失、遭窃、刊登在报纸上或其它方式泄露，所有的安全性就不复存在。

密钥泄露并不需要做什么。理想的情况下，用户知道他们的密钥何时泄露。如果由 KDC 管理密钥，用户应及时申报他们的密钥已泄露。如果没有 KDC，用户应及时通知接收信息的所有客户。用户应该公布密钥泄露后任何接收到的信息都是可疑的，大家不要再使用该密钥（或相关的公钥）传送信息。系统应该使用时戳技术，这样用户就可推测出哪个信息是正常的和哪个信息是可疑的。

如果用户并不知道到底何时泄露密钥，问题就困难得多了。他们得收回契约，因为密钥得到者可以冒充签署。如果放之任之，那么人人在签署之前都可声明密钥已泄露而收回契约，这只好让仲裁人员来决定了。

私钥泄露一般要比密钥泄露更严重，因为私钥更为永久性。密钥定期更换，而公钥/私钥对并非如此。如果 B 得到了 A 的私钥，他就可以网络上冒充 A 阅读加密信件、签署信件和契约合同等等。

私钥泄露的信息快速在网络上传播是很严重的，必须马上通知所有的公钥数据库某个私钥已泄露，以免意想不到的人用泄露的密钥加密信息。

这种措施和提示并非十分完美，但这也许是最佳解决方案。这类事件的教训是保护好密钥，尤其要保护好私钥。

7.2.8 密钥生存期

没有一个加密密钥可以无限期使用，这有许多理由：

1. 密钥使用越久，泄露的机会就越多。一个密钥使用一年的泄露机会要比使用一天大得多。

2. 密钥使用越久，一旦泄露遭受的损失就越大。如果密钥只使用于加密文件服务器上的单个预算文档，那么密钥泄露只是意味着该文档泄露，如果加密文件服务器上所有预算信息都采用同一密钥，则密钥泄露时的损失就严重得多了。

3. 密钥使用越久，对攻击者去破译（即使是蛮力攻击）的诱惑就越大。破译了两个军事部门某天共同的密钥就使攻击者那天可在两个部门之间阅读和伪造信息。破译了整个军事指挥机构一年内公用的密钥就使攻击者全年为所欲为。

4. 一般而言，用同一个密钥加密的密文越多，密码破译会越容易。

对于任何密码应用场合，必须确定密钥的生存期。不同密钥的生存期也许不同，通讯会话密钥的生存期相对较短，取决于数据的价值和在给定时期内的加密数据量。1Gbit/s 的通讯链路上的密钥要比 9600Baud 调制解调器链路上的密钥更换得更频繁一些。假如新密钥传送过程中用主钥加密，会话密钥应至少每天更换。

另一方面，主钥无需如此频繁地更换，只有在密钥更换时偶尔用到主钥(大约每天一次)，密码分析者能用的密文只有一点点。然而，如果主钥泄露，其可能造成的损失是巨大的：所有通讯用各个密钥加密，而密钥又以主钥加密。有些应用场合中主钥仅每月或每年更换一次。

用于加密要保存的数据文件的密钥不能经常更换。文件可能以加密的形式在磁盘上保存数月甚至数年之久，直到有人需要重新使用它们。每天用一个新密钥解密和重新加密并不会提高安全性，反而会给密码分析者提供更多的参考资料。一种解决方法是对每个文件用唯一的密钥加密，再将所有密钥用主钥加密。主钥应妥善安全保管，当然，主钥丢失意味着丢失所有的文件。

公钥密码系统中的私钥的生存期因具体应用场合而异。用于数字签名和身份认证的私钥可持续数年之久(甚至整个生存期)，用于掷币协议的私钥可在协议结束后马上丢弃。即使希望密钥的安全性持续整个生存期，还是应该谨慎地每二、三年更换一次。在多数公钥密码网络中的私钥只能胜任二年：到时用户得换一个新的私钥。原来的密钥也得妥善保管，万一用户需要验证来自更换前的签名。但新文档得用新密钥签署，以减少攻击者所拥有的签了名的文档数量。

7.2.9 密钥销毁

假定密钥是定期更换的，原密钥就必须销毁。原密钥即使再也用不到也还是有价值的，因为攻击者可以用原密钥破译用原密钥加密的老信息。

密钥必须安全销毁。如果密钥写在纸上，应把纸粉碎或焚烧。对于耗费数百万美元、花费百万年的蛮力攻击，本书中的算法是安全的，但如果攻击者从你的垃圾筒里取走碎片，然后设法将其拼合在一起而找到你的密钥，他所化的代价会小得多。

如果密钥写在 EEPROM 硬件上，应该对其作多次覆盖写。如果密钥写在 EPROM 或 PROM 中，应把芯片碾碎并撒入狂风中。如果密钥存储在计算机磁盘上，也应对该存储区域作多次覆盖写。

潜在的问题是：计算机里密钥可以复制并存储多个副本。如果计算机由自身管理存储器，并经常把程序交换进/出存储器，则该问题显得特别严重。无法保证计算机里的密钥已经彻底删除，尤其如果计算机操作系统控制删除操作。偏执的用户应考虑自己编写一个特殊的删除程序，在未被使用的存储块中搜索密钥位模式的副本，再删除这些存储块。如果需要也删除“交换”文件的内容。

7.3 公钥管理

公钥密码系统简化了密钥管理，但也有其独特的问题。不论网络上有多少用户，人人拥有一个私钥。如果 A 要给 B 传送信息，她得有 B 的公钥。获取方法有：

- 她可向 B 索取
- 她可从集中数据库中获取
- 她可从自己的私人数据库中获取

以前曾以攻击者替换 B 的密钥为例讨论了对公钥密码系统的许多可能攻击方法。情况是 A 要给 B 传送信息，她访问公钥数据库获取了 B 的公钥，但是狡猾的攻击者把他的密钥替换成了 B 的密钥(如果 A 直接向 B 索取，攻击者截取 B 的传输并把他的密钥替换 B 的密钥)。A 用攻击者的密钥加密报文并传送给 B，攻击者截取了报文并解密，他用 B 的真正密钥重新加密

再传送给 B，A 和 B 对此都一无所知。

公钥认证

公钥认证是由可靠的人签署的某人的公钥，认证用于阻止密钥替换。在公钥数据库中，B 的认证包含的信息比他的公钥还要多得多：有关 B 的信息（姓名、地址等）以及由 A 信任的某人（通常被认为是认证权威(certifying authority)）签了名。通过对密钥和有关 B 的信息签名，认证人证明有关 B 的信息是正确的和公钥是属于 B 的。A 核实签名，接着她可以放心地使用公钥，因为她知道这密钥的确是 B 的。认证在许多公钥协议中起着重要的作用。

分布式密钥管理

在有的情况下，集中式密钥管理并不可行。也许没有 A 和 B 都信赖的认证权威，也许 A 和 B 只信赖他们的朋友，也许 A 和 B 对谁也不信赖。

公用的"密佳"(PGP)程序（无版权），采用分布式密钥管理，以介绍人(introducer)解决了这个问题。介绍人是系统中的其他用户，他们在朋友们的公钥上都留有签名。例如，当 B 生成他的公钥，他把副本交给他的朋友 C 和 D，他们认识 B，他们在 B 的密钥上都签了名并把签名的副本交给 B。于是，当 B 把他的密钥交给陌生人 A 时，他随呈上二个介绍人的签名，如果 A 也认识并信赖 C，她就相信 B 的密钥是正确的。如果她对 C 和 D 都只是认识一点点，她也会相信 B 的密钥是正确的。如果她对 C 和 D 都不认识，她就不会信赖 B 的密钥。

日积月累，B 会有许许多多介绍人。如果 A 和 B 涉足类似的生活圈，A 认识其中一位 B 的介绍人的机会是很大的。

为防止攻击者替换密钥，介绍人在签名之前必须确信 B 的密钥是属于 B 的，也许介绍人要求面对面交接密钥或通过电话验证。

这种方法的优点是无需人人都信赖的 KDC，缺点是当 A 收到 B 的公钥时，不能保证她认识某个介绍人，所以也不能保证她信赖密钥的正确性。

第8章 常用算法

8.1 块密码方式

各种块(分组)算法均以块为单位(通常为64位)处理明文和密文。仅仅在程序中置以某一块算法并让它运行是远远不够的,用户应根据不同的需求以不同的方式使用块算法。

8.1.1 电码本方式

使用块密码最明了的方式是把各块明文加密成相应密文块,这称为电码本(Electronic Codebook,ECB)方式,因为一块明文总是加密成同一块密文。块的典型大小为64(bit),于是电码本有2^{64}项,这个数字太大而难以从中预测,并且对每个不同的密钥,电码本是不同的。

电码本方式是块密码的基本工作方式,其优点是每个明文块独立加密,不必连续地加密文件,可以首先加密中间的10块,再是末尾块,最后加密起始块。可以随机地访问加密文件(如数据库)是很重要的,如果数据库以ECB方式加密,则任何记录都可以独立解密,也可以在数据库中间添加或删除记录(假设一条记录由若干加密块组成)。

ECB方式的缺点是如果密码分析者掌握多个报文的明文和密文,无需知道密钥就可编译电码本。现实中许多情况,报文段往往趋于重复,一个报文与另一个报文有共同的位序列。计算机生成的报文(如E-Mail)会有规则的结构,报文中可能含有价值不大的参量以及0和空格的长串,其它计算机生成的报文也许在同一地方有重复数据。

如果密码分析者知道明文块“5e081bc5”加密成密文块“7ea593a4”,无论该密文块出现在何处都可马上解密出来。如果应用程序加密报文带有许多冗余数据,则这些冗余数据出现在报文中的同一地方,可引发强有力的攻击。

在报文的起始和末尾处最为脆弱,因为这些地方的头和尾确定地包含着发送者、接收者和数据等的信息,解决这种被称为固定(stereotyped)起始和固定末尾的问题,需要采用另外的块加密方式。

8.1.2 块重发

ECB中另一个潜在的并更为严重的问题是敌手无需知道密钥(或算法细节)就可修改加密报文,以此欺骗原接收者。

为说明这个问题,考虑银行之间的转帐系统。银行经常要在不同银行的帐号之间转帐,为简化不同计算机系统之间的操作,银行网络转帐系统约定采用如下的标准报文格式:

付方银行名	12字节
收方银行名	12字节
储户名	48字节
储户帐号	16字节
转帐金额	8字节

当然这个报文是用 ECB 方式块算法加密的。

罪犯欲通过对两家银行之间的通讯信道搭线窃听其中的信息而“致富”，首先他的计算机记录下银行 1 到银行 2 的所有加密报文，接着他从银行 1 将 100 元转帐到银行 2 他的帐号上，后来再做一次相同的转帐。利用计算机可以在记录下来的各条报文中找出相同的报文对，这个报文实现把 100 元转帐到他的帐号上。如果相同的报文不止两个(当然也有其他人在这两个银行之间转帐)，他就再做一次转帐并记录结果。最后，他可找出实现他自己转帐的报文。

现在他可随意把该报文发送到通讯线上，他每给银行 2 发送一次这段报文，他的帐号上就会增加 100 元，当这两家银行核对转帐时(也许在每天凌晨)时，他们会注意到这个“幽灵”的转帐经过，但如果罪犯十分狡猾，他已经提了款并逃往国外。这类罪犯转帐金额可远远不止 100 元，也可对许多家银行都转了帐。

简单的防御措施，银行可以在他们的报文中加以时戳(timestamp)标记：

日期/时戳	1 块
付方银行名	1.5 块
收方银行名	1.5 块
储户名	6 块
储户帐号	2 块
转帐金额	1 块

这样可以容易辨识出两个相同的报文而防伪造。但罪犯还可通过称为块重发(block replay)技术进行攻击。假设密码算法的块大小为 64 位(DES 和其它许多算法的标准)，记录将以 13 块的形式加密，如图 8.1 所示。由图中看出，罪犯可以抽取出关于 A 的姓名和帐号的 8 个密文块(第 5～12 块)。

1	2	3	4	5…10	11	12	13
时戳	付方银行		收方银行	储户名	帐号		金额

图 8.1　加密块

罪犯随意截取银行 1 到银行 2 的报文，并把其中第 5～12 块替换成自己的姓名和帐号，再把报文发送给银行 2。他不需要知道原储户是谁，也不需要知道转帐金额多少(虽然他可以根据篡改的报文与帐号里余额之间的对应关系确定加密块的转帐金额)，他只要把姓名和帐号改成自己的再看看自己帐号上的增长额就行了。

银行绝非一天就能发觉这个情况的。当他们核对转帐时，一切都相符，也许直到原储户发觉钱并没有到达他的帐号，这件事永远不会得以发现。

银行可通过经常更换密钥把由此造成的损失减小到最小程度，但这也只不过是意味着罪犯得加快作案速度。这是 ECB 块密码方式的基本问题：因为各块明文都是独立处理的，易受块重发的攻击。其解决方法是用链接(chaining)技术。

8.1.3　密码块链接方式

链接采用反馈方法，前块的加密结果反馈到当前块的加密过程中，即前块用于修改后续块的加密过程。每个密文块不仅取决于与之对应的明文块，还与前面所有的明文块有关。

在密码块链接(Cipher Block Chaining,CBC)方式中,明文在加密之前与前面密文块异或。图 8.2 说明了块链接过程。明文块加密之后,其结果密文也存储在一个反馈寄存器中,在加密下一个明文块之前,它与反馈寄存器异或,异或结果再作为加密例程的下一个输入。其结果密文又存入反馈寄存器中,用于与下一个明文块异或。如此重复直到报文结束。每一个明文块的加密过程都依赖于前面的所有块。

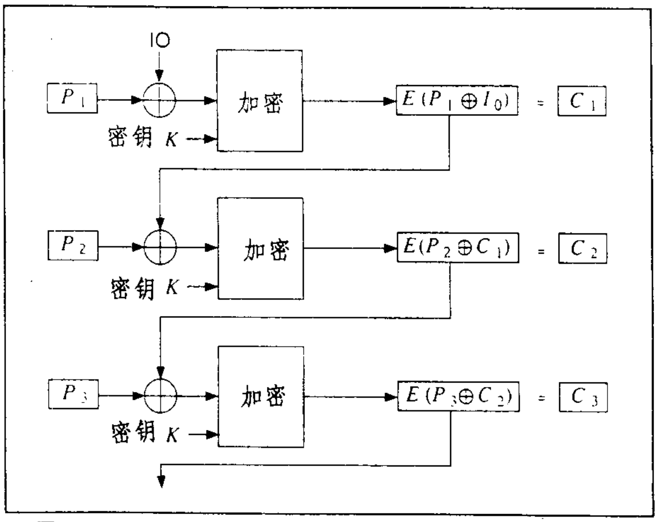


图 8.2　密码块链接方式

解密过程是很简单的。密文块按常规解密,并也存入反馈寄存器中,当下一密文块解密以后,它与反馈寄存器异或,并把下一密文块存入反馈寄存器。如此重复直到报文结束。

其数学表示为:

$C_i = E_k(P_i \text{XOR } C_{i-1})$

$P_i = C_{i-1} \text{XOR } D_k(C_i)$

初始化矢量

CBC 使相同的明文块加密成不同的密文块(若前面的明文块不同时),但二个相同的报文还是可能加密成同样的密文,更糟的是,开头相同的二个报文将以同样的方式加密,直到二者相异。有些报文有一个共同的报头:如页面台头或"From"行等等。虽然不可能受块重发攻击,但是这种相同的开头会给密码分析者提供有用的信息。

防止的方法是用随机数作为加密的第一块,这个随机数块称为初始化矢量(Initialization Vector,IV)、初始化变量(initializing variable)或初始链接值(initial chaining value)。IV 没有实际意义,它仅仅使每个报文是唯一的。无需将 IV 保密,当接收者对此块解密后,仅将之用于填充反馈寄存器,否则可以不加理会。

IV 应是一个随机数。它可以是编号,在每个报文之后递增并且在密钥生存期内不会重复。对于储存数据加密,它可以是用于检索数据的索引的函数。

有了 IV,即使相同的明文块也加密成不同的密文数据,这样,窃听者将不可能从密文中得到有关明文的信息。

IV 不需保密并可以与密文一起传送,然而,对每个发送报文都得更换 IV。

填充技术

大多数报文不能恰好都划分成 64 位(或其它大小)的加密块,最后一块往往是个短块。对

此有二种解决方法。

最容易的方法是用一些规则的字符模式，如用 0，1，或交替的 0 和 1 填充短块使其完整。如果希望应用程序能够在解密之后删除填充数据，则在报文的最后一个字节中添加填充的字节数。例如，假设块大小为 64 位，最后一个短块为 3 字节(24 位)，需要填充 40 位(5 字节)：4 字节的 0 和最后一字节 5，它表示填充字节数为 5。解密之后，程序就知道删除最后一块中的最后 5 个字节。也可用文件结束符(EOF)指示明文的最后一个字节。

有时要求密文与明文完全一样大小，如密文要严格地在同一存储区中替换加密之前的明文。在这种情况下，对最后一短块的加密要特殊处理。假设短块为 j 位，加密最后一整块之后，再对同一密文加密，选择加密密文的最左边 j 位与短块异或生成密文。如图 8.3 所示。

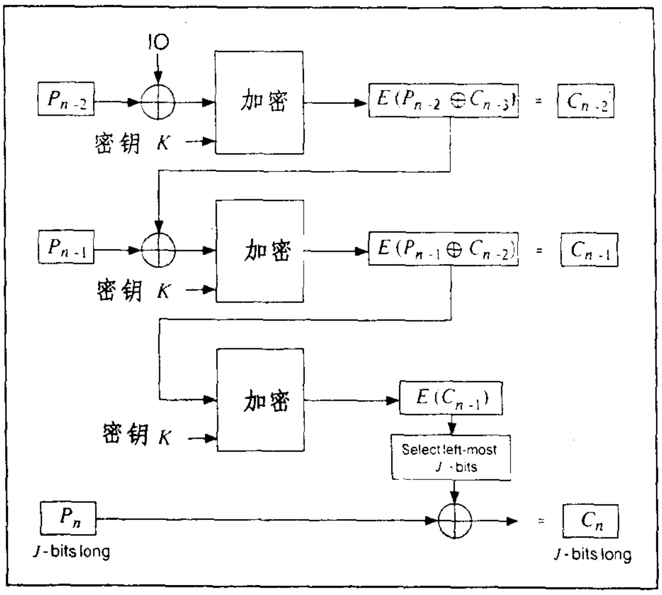


图 8.3　加密最后一块的方法

这种方法的缺点是当攻击者不能恢复最后一个明文块时，他可以有计划地改变密文中的个别位，如果密文的最后几位含有重要信息，这将发生问题。如果最后几位简单地含有校验和或其他琐碎信息，则无关紧要。

错误扩散

CBC 以在加密端的密文反馈(feedback)和解密端的密文前馈(feedforward)为特征，这意味着错误必定连带一起处理。明文块中的一个错误将影响密文块以及所有后续密文块。但这并不重要，因为解密时会把影响逆转过来，恢复的明文中同样只有一个错误。

密文中的错误更为常见。通讯线路中的噪声和存储介质的故障都会导致密文错误。CBC 方式中，密文中的一位错误影响解密后明文中的一块和一位。含有错误的一块是完全被扭曲的，后续块也有一位错误，它与密文的错误处于同一位置。

错误不会影响第二块以后的各块，这种 CBC 是具有自恢复功能的。一个错误影响二块，但系统自行恢复并继续对所有后续块正确处理。

虽然 CBC 可以迅速恢复位错误，但它不能恢复同步错误。如果密文流(序列)中增加或丢失了一位，则所有后续块都错位了一位，解密操作将无限制地生成错误的明文。无论是采用帧还是采用以多重块大小存储数据的手段，所有使用 CBC 的密码系统都必须保证块结构保持完整。

密文中发生的一个小错误会导致大量的明文出错，这一特性称为错误扩散，这是该方法的主要问题。为防止错误扩散，在加密之后和解密之前，通信线路上应运用错误检测和恢复设施（如图 8.4 所示）。如果通信线路上有噪音，解密将致使噪声更加严重。

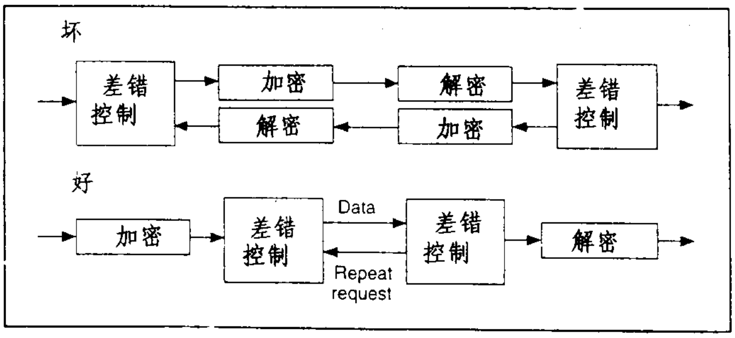


图 8.4 加密的错误检测

8.1.4 密码反馈方式

CBC 方式要等到接收到数据的一个完整块以后才能开始加密，这在某些网络应用中是个问题。例如，在安全网络环境下，用户在终端上键入的每个字符必须马上传送给主机。当数据以字节为单位处理时，CBC 方式是不适合的。

密码反馈（Cipher Feedback，CFB）方式加密数据的单位比块小得多。本例每次加密一个 ASCII 字符（称为 8 位 CFB），当然也可以用 1 位 CFB 每次加密 1 位数据。

图 8.5 说明对 64 位数据处理的 8 位 CFB 算法。CFB 方式的块算法施加操作于输入块大小（图中为 64 位）的队列上，队列类似于 CBC 方式最初用一个初始化矢量填充。这队列被加密，加密结果的最左边 8 位与明文的第一个 8 位字符异或，生成密文的第一个 8 位字符。传输这加密后的字符，并将其移入队列的最右边 8 位的位置，所有其它位左移 8 位，原最左边的 8 位丢弃。接着以同样的方式加密下一个字符。解密是加密的逆过程，加密端和解密端双方的块算法都以加密方式工作。

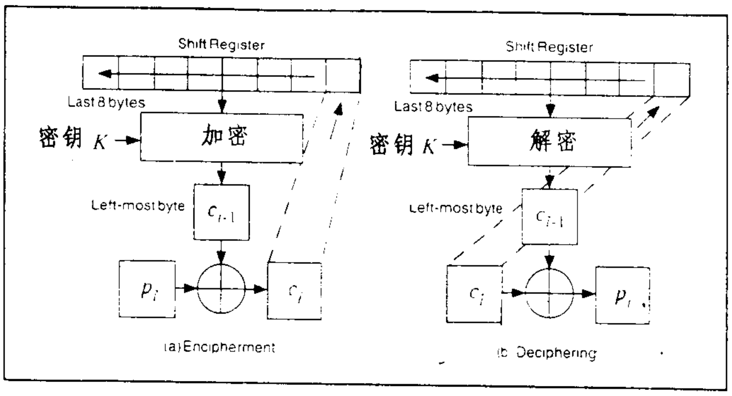


图 8.5 8 位密码反馈方式

类似于 CBC，CFB 也把明文字符连接在一起，所以密文依赖于所有前面的明文。

初始化矢量

初始化 CFB 操作时，移位寄存器必须以 IV 填充。在密钥的生存期内对每一个报文要采用

不同的 IV。然而，IV 并没有保密价值，它可以以明文形式传送。

错误扩散

类似于 CBC，CFB 方式中明文中的错误会影响所有后续密文，同样在解密时可把错误逆转过来。

密文中的错误更有趣。密文中一位错误的首次影响是引起明文中一位错误，在该错误进入移位寄存器后，它篡改密文直到离开寄存器的另一端。在 8 位 CFB 中，一位错误会篡改 9 字节密文，在此之后系统自行恢复，所有后续密文都正确地解密。

这种错误扩散的一个微妙问题是如果攻击者知道传送的明文，他可在给定块中改正一些位并将该块解密得到他所希望的结果，虽然下一块解密后仍有错误，但是危险已经产生了。

CFB 对于同步错误也是自恢复的。错误进入移位寄存器，它篡改 8 字节数据直到离开寄存器的另一端。

8.1.5 输出反馈方式

输出反馈（Output Feedback，OFB）方式类似于密码反馈方式，其区别只是前面输出块的部分移入队列的最右边（如图 8.6 所示）。解密是加密的逆过程，加密端和解密端双方的块算法都以加密方式工作。这一方法有时称为内部反馈（internal feedback），因为反馈机制是独立于明文和密文流（序列）的。

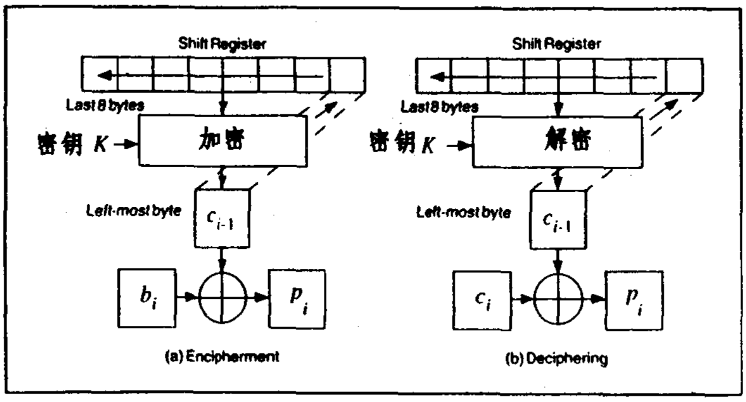


图 8.6 输出反馈方式

初始化矢量

OFB 的移位寄存器最初装入 IV。在密钥的生存期内对每一个报文要采用不同的 IV。IV 可以以明文形式传送。

错误扩散

OFB 的优点是没有错误扩散，密文中的一位错误引起解密后的明文中的一位错误，这在一些数字化的模拟量传输（如数字化的音频和视频数据）中很有用，因为这些场合可以允许一位错误而不能允许错误扩散。

另一方面，丢失同步是一个很严重的问题。如果加密端和解密端的移位寄存器不同，恢复的明文将是扭曲的。采用 OFB 方式的所有系统必须有检测同步丢失的机制和为恢复同步而在二个移位寄存器中填充新的（或同一个）IV 的机制。

OFB 方式的安全性

Gait 对 OFB 的分析表明只有当反馈大小等于块大小时才能采用 OFB。例如，在 64 位的 OFB 中只能采用 64 位算法，尽管美国政府认可 DES 的其它反馈可能性，但应避免使用。

8.1.6 计数器方式

计数器方式是使用序列数的一种类似于 OFB 的方式，用于填充寄存器的不是加密算法的输出，而是计数器，每一个加密块之后，计数器递增某个常量(通常为 1)。同步和错误扩散特性与 OFB 相同。

8.1.7 其它方式

块链接方式

块链接(Block Chaining,BC)方式使用块算法，只是简单地将块密码的输入与前面所有密文块的异或结果异或。类似于 CBC 方式，操作开始时用一个初始化矢量 I0。

其数学表示为：

$$C_i = E_k(P_i \text{ XOR } F_i) \qquad F_{i+1} = F_i \text{ XOR } C_i$$

$$P_i = F_i \text{ XOR } D_k(C_i) \qquad F_{i+1} = F_i \text{ XOR } C_i$$

与 CBC 方式一样，BC 方式的反馈操作也会在明文中扩散错误。使用 BC 方式的主要问题是由于密文块的解密依赖于所有前面的密文块，密文块中的一个错误会导致所有后续密文块的解密不正确。

传播密码块链接方式

传播密码块链接(Propagating Cipher Block Chaining, PCBC)方式类似于 CBC 方式，只是在加密之前(或解密之后)前面的明文块和前面的密文块都是与当前明文块异或(如图 8.7)。

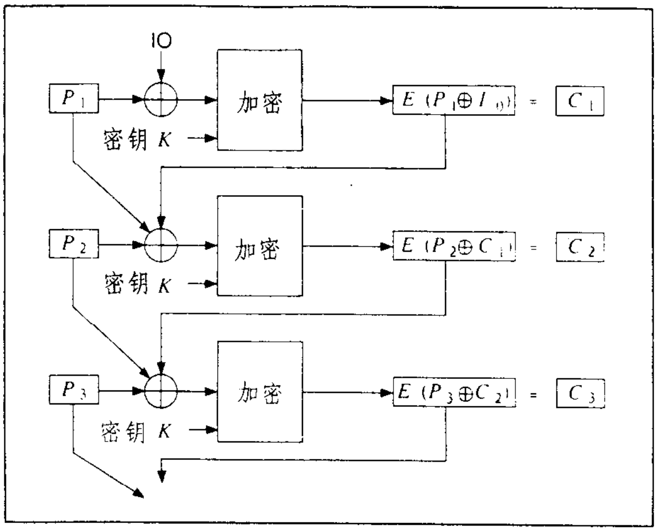


图 8.7 扩散密码块链接方式

$$C_i = E_k(P_i \text{ XOR } C_{i-1} \text{ XOR } P_{i-1})$$

$$P_i = C_{i-1} \text{ XOR } P_{i-1} \text{ XOR } D_k(C_i)$$

PCBC 方式用在 Kerberos 版本 4 中完成一遍的加密和完整性检验。PCBC 方式中，密文中的错误会导致所有后续块的解密不正确，这意味着在报文末尾检验一个标准块将保证整个报文的完整性。

然而这种方式有一个问题：交换二个密文块将导致这二个相应明文块的解密不正确，但由

于明文和密文异或的性质，这个错误不会被发觉。所以如果仅对最后几个加密的明文块检验完整性，则接收了部分被篡改的报文也不会发觉。它曾用于 Kerberos 版本 4 中，但发现了上述缺点之后在版本 5 中被 CBC 方式所替代。

其它方式

其它还有一些没有广泛使用的方式。明文块链接(Plaintext Block Chaining, PBC)方式类似于 CBC，只是前面的明文块与明文块异或而不是与密文块异或。明文反馈(Plaintext Feed-back,PFB)方式类似于 CFB，只是填充移位寄存器是用明文而不是密文。有的方式在数据路径和反馈路径中都做加密操作。其它还有 CBCPD 和 OFBNLF 方式。

如果密码分析者有密钥搜索工具，则如果他们能猜中明文中的一块，他们就可找到密钥。

8.1.8 密码方式选择

如果简单明了和速度是主要因素，ECB 是使用块密码最容易也是最快的方式，但同时也是最脆弱的。除了易受重发攻击外，ECB 方式算法的密码破译也是最容易的。

可以根据具体需求选择 CBC、CFB 或 OFB 方式。CBC 方式往往最适合于文件加密，它只比 ECB 方式稍稍慢一点，唯一的区别是一个和块大小完全相同的寄存器存储空间和每块的异或操作。但明显地提高了安全性，虽然存储数据中有时会有位错误，但几乎不会有同步错误。如果用户是基于软件实现，CBC 是最佳选择。

OFB 和 CFB 要比 CBC 方式慢得多，因为它们的块算法中每次迭代只加密少量的明文。如果用户允许错误扩散而换取自行恢复同步丢失的能力，可采用 CFB 方式。如果用户不能处理任何的错误扩散，可采用 OFB 方式。

CFB 往往用于加密字符流，并且要求象终端和主机之间的连接那样独立处理每个字符。OFB 最常用于不允许错误扩散的高速同步系统。

请不要轻易采用“离奇”的方式。四种基本方式 ECB、CBC、OFB 和 CFB 中的任何一种均能满足需求，这些方式并非十分复杂并且也不会降低系统的安全性。也许某种复杂的方式能提高系统的安全性，但往往仅仅增加了复杂性而已。没有一种离奇的方式具有更好的错误扩散和错误恢复特性。

8.2 多重加密

对同一明文块作多次加密称为多重加密。以同一算法和同一密钥对同一明文加密二次并不改变蛮力搜索的难度，需采用多个密钥，才能增强安全性。

8.2.1 双重加密

提高块算法安全性的一种简单方法是以二个不同的密钥执行二次加密，第一次用第一个密钥 K_1 加密，第二次用第二个密钥 K_2 加密。解密是其逆过程。

$$C=E_{k2}(E_{k1}(P))$$

$$P=D_{k1}(D_{k2}(C))$$

如果块算法不是群，则不一定存在第三个密钥 K_3，使得：

$$C=E_{k2}(E_{k1}(P))=E_{k3}(P)$$

这样，用穷举搜索法攻击双重加密产生的密文块要困难得多，不再是需要测试 2^n 次(n 为

钥长)，而是 2^{2n}次。如果算法是 64 位的，找到双重加密密文的密钥要测试 2^{128}次：似乎毕生也做不完。

但这个结论是不正确的。Merkle 和 Hellman 采用已知明文攻击设计的时空折换方法可以在 2^{n+1}次而不是 2^{2n}次测试内攻破双重加密(他们以 DES 说明，但结果可推广到所有的块算法)。这种攻击称为中间会合攻击，它从一端做加密，从另一端做解密，在中间比较结果。

这种攻击以如下方式进行：密码分析者知道 P_1，C_1，P_2 和 C_2，即

$$C_1 = E_{k2}(E_{k1}(P_1))$$

$$C_2 = E_{k2}(E_{k1}(P_2))$$

对每一个可能的密钥 K(无论是 K_1 还是 K_2)，计算 $E_k(P_1)$并把结果存储在存储器中，做完以后，对每个 K 计算 $D_k(C_1)$并在存储器中查找相同的结果，如果找到，则可能是当前密钥为 K_2 以及存储器中的密钥为 K_1，试着以 K_1 和 K_2 加密 P_2，如果得到了 C_2 则可相当确信(成功的概率为 2^{2n-2b}，其中 b 为块大小)找到了密钥。如果没有找到则继续搜索。所需测试次数的期望值为 2^{2n-2b}，所需执行的加密测试的最大值为 2^{n+1}。如果错误的可能性太大，可以使用第三个密文块而使成功的概率达到 2^{2n-3b}。还有其它的优化措施。

这种攻击需要大量的存储空间：2^n 块。对于 56 比特的算法，这将是 2^{56}个 64 比特的块，即 2^{59}字节。目前的存储器是完全不能胜任的，这足以说明攻击双重加密简直是妄想。

如果块密码是群，分别以 K_1 和 K_2 做二次加密等效于以第三个密钥 K_3 加密，这并没有提高安全性。但证明一个加密算法不是群是非常困难的，密码编制人员努力试图设计不是群的算法。例如 DES 就不是群。

8.2.2 三重加密

由 Tuchman 提出的一个更好的方法是以二个密钥执行三次加密：第一次用第一个密钥 K_1，第二次用第二个密钥 K_2，最后再用 K_1。他建议发送者先以 K_1 加密，再用 K_2 解密，最后用 K_1 加密；接收者先以 K_1 解密，再用 K_2 加密，最后用 K_1 解密。有时也称这种方法为加密-解密-加密(encrypt-decrypt-encrypt，EDE)方式。

$$C = E_{k1}(D_{k2}(E_{k1}(P)))$$

$$P = D_{k1}(E_{k2}(D_{k1}(C)))$$

如果块算法的密钥为 n 位，则 EDE 的密钥为 2n 位。IBM 设计了这种独特的加密-解密-加密方法，用于保持与算法的常规实现的兼容性：设置二个密钥相同就相当于以每密钥各加密一次。EDE 并没有内在的安全性，但在 X9.17 和 ISO 8732 标准中已采用 EDE 来改进 DES 算法。

EDE 不易受中间会合攻击，但 Merkle 和 Hellman 设计的另一种时空折换方法可以在 2^{n-1}步内攻破 EDE，用了 2^n 块存储器。这是一种选择性明文攻击，需要大量的已选明文。Paul van Oorschot 和 M. Wiener 把它转化成已知明文攻击，比对二个密钥的蛮力搜索所需的时间减少了 4 个数量级(2^{2n-4})。

超乎寻常考虑时，推荐使用三个不同密钥执行三次加密，钥长是增加了，但存储密钥一般并不成问题。

$$C = E_{k3}(D_{k2}(E_{k1}(P)))$$

$$P = D_{k1}(E_{k2}(D_{k3}(C)))$$

最佳的时空折换攻击法需要 2^{2n}步和 2^n 块存储器。将三个密钥设置成相互一样时，即与单

次加密兼容。

以三个不同密钥的三重加密的安全性与双重加密相差无几。

8.2.3 带填充的三重加密

如果报文较长(比若干加密块长得多),可以用填充的手段进一步改进三重加密。把一串随机位(长度为半块)填充进第一次和第二次、第二次与第三次加密之间(见图 8.8)。

$$C=E_{k3}(PAD(D_{k2}(PAD(E_{k1}(P)))))$$

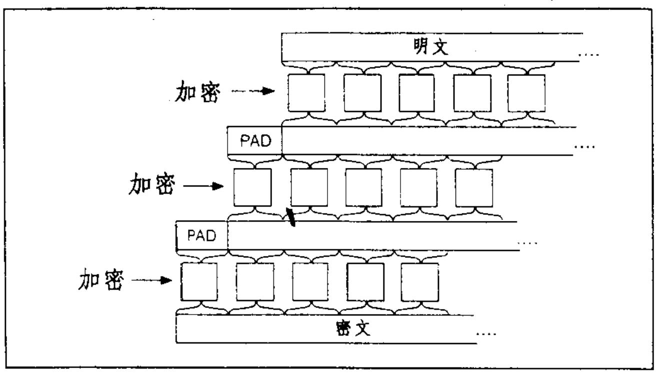


图 8.8 带填充的三重加密

填充不仅打乱了数据模式,而且把加密块像积木一样叠加。

8.2.4 块长加倍

R. Outerbridge 提出把使用多重加密的算法的块长加倍。在实现之前,首先看看中间会合攻击者的可能性。图 8.9 所示的方法类似于前面提到的方法是易受中间会合攻击的,图 8.10 所示的 Outerbridge 的方法是安全的。

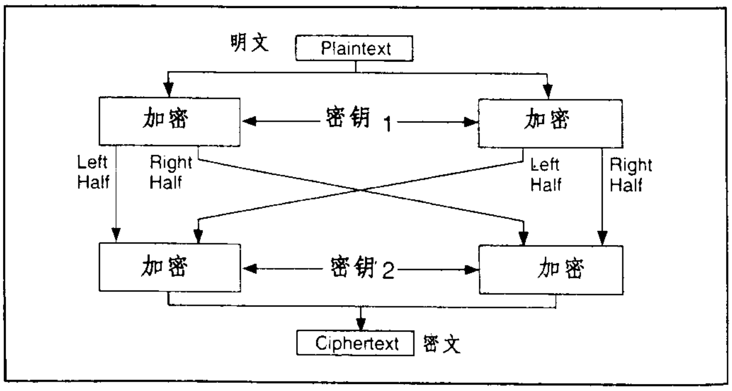


图 8.9 块长加倍易受攻击

8.2.5 以多个算法多重加密

先用算法 A 和密钥 K_A 加密一次,再用算法 B 和密钥 K_3 加密一次将会如何?除易受同样的中间会合攻击之外,并没有保证这二个算法能协作提高安全性,也许这二个算法交互作用反而会降低安全性。甚至以二个或三个不同的算法作三重加密也不会象想象中那么安全,密码术是精巧的艺术。

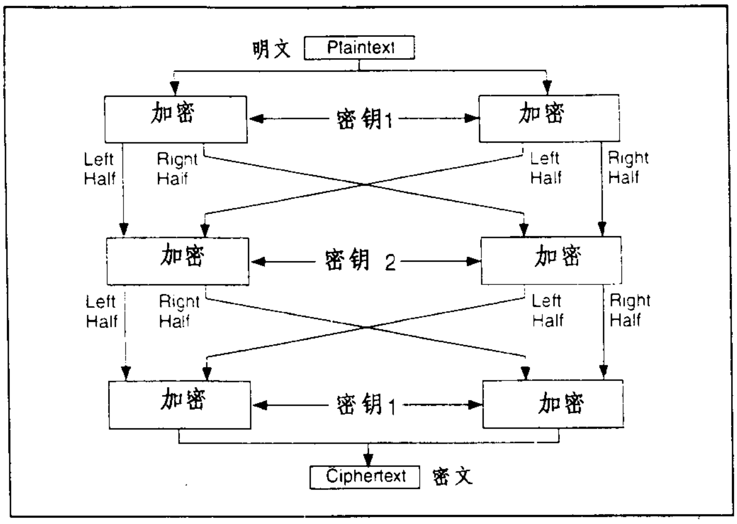


图 8.10 正确的块长加倍方法

8.3 流(序列)密码

流密码(stream cipher)是逐位将明文转换为密文的算法，流密码的最简单实现如图 8.11 所示。密钥流发生器输出位流：$K_1,K_2,K_3,\cdots,K_i$，该位流与明文位流 $P_1,P_2,P_3,\cdots,P_i$ 异或生成密文位流。

$$C_i=P_i\oplus K_i$$

在解密端，密文位流与同一个位流异或恢复明文位流：

$$P_i=C_i\oplus K_i$$

这是由于

$$P_i\oplus Ki\oplus K_i=P_i$$

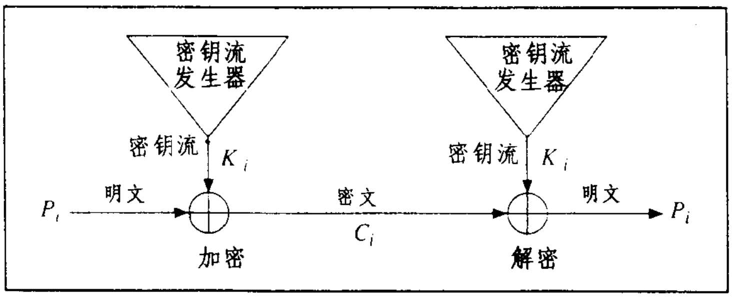


图 8.11 流密码

系统的安全性完全取决于密钥流发生器的内部。如果密钥流发生器输出恒为 0 的流，密文将与明文完全一致，整个操作过程毫无价值。如果密钥流发生器输出重复的 16 位模式，算法成为几乎没有安全性的简单异或操作。如果密钥流发生器输出无穷的随机(不是伪随机，而是真正随机)位流，则是“一次一密”(one-time pad)密码，具有极好的安全性。

现实性的流密码在于简单的异或和一次一密之间——理想上更接近于一次一密。密钥流发生器输出一个看似随机的位流，但其实是个在解密时可以完全再生的确定流。密钥流发生器的输出越接近随机，密码分析者要攻破它就越困难，但是设计一个输出随机的密钥流发生器并

非易事。

流密码除输出外还有许多问题。如果密钥流发生器每次输出同样的位流，得到的密码系统是易于攻破的。下例说明其原因。

如果明文报文为

10001101

密钥流发生器输出的密钥流为

01100111

则结果密文为

11101010

E 有一个密文的副本并且设法获得了明文，她只要将明文和密文异或即可得到密钥流。或者，如果她有以同一个密钥流加密的二个不同密文，她把它们异或在一起就可得到二个相互异或的明文报文。这是易于攻破的，她可将其中一个明文与密文异或就可得到密钥流。

于是，无论何时来了另外的密文报文，它有了解密必需的密钥流。另外，她也可以解密以前截取的所有密文。当 E 得到一个明文/密文对时，她可以获得一切。

这就是为什么所有的流密码都有密钥的原因。图 8.12 说明密钥如何添加到系统中。密钥流发生器的输出是密钥的函数，用这个密钥，密钥流发生器可能输出位流“11001110”，于是，如果 E 得到一个明文/密文对，她只能解开以该密钥加密的报文。更换了密钥，敌手只能从头再来。

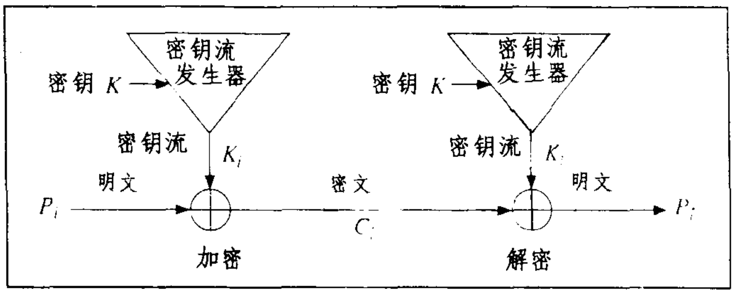


图 8.12　带密钥的流密码

由于对每个报文得更换密钥，流密码往往不适合用于加密离散的报文，而更适用于加密通讯业务的无穷流：例如二台计算机之间的 T-1 连接。

由于密钥流发生器在加密端和解密端必须产生同样的输出，故必须是确定性的。由于它以有限状态机（计算机）实现，序列最终总归重复。这种密钥流发生器称为周期性的。除了一次一密，所有的密钥流发生器都是周期性的。

密钥流发生器的一个重要设计目标是周期要长，比更换密钥之间产生的输出的位数要长得多。加密一个连续的 T-1 连接的密钥流发生器每天要加密 1011 位数据，即使密钥每天更换，密钥流发生器的周期必须大于该数量级，如果周期足够长，密钥只需每周甚至每月更换一次。

密钥流发生器由三个基本部分构成（见图 8.13）。内部状态描述密钥流发生器的当前状态，密钥和内部状态均相同的二个密钥流发生器输出相同的密钥流。输出函数根据内部状态产生密钥流的位输出。下一状态函数根据内部状态产生新的内部状态。

设计一个好的密钥流发生器是不容易的，过去几年已经做了大量的工作。

8.3.1 同步流密码

在同步流密码中，密钥流的产生是独立于报文流的，如图 8.12 所示。在加密端，密钥流发生器逐位输出密钥流，在解密端，另一个密钥流发生器逐位输出相同的密钥流，只要二个密钥流发生器同步，整个过程就能保持正常。如果其中某个跳过了一个周期或者传输中丢失了某位密文，则自从错误点之后的所有密文的解密都是不正确的。

如果这种情况发生，发送者和接收者得在处理之前使二个密钥流发生器重新同步。更糟糕的是，他们必须做到为保证密钥流没有任何重复而使二个密钥流发生器重新同步，所以重新把密钥流发生器设置到前面某个状态而明显是不行的。

另外，同步密码不会扩散传输错误。如果传输过程中篡改了某位（比某位完全丢失更有可能），只有篡改的位的解密是不正确的，所有先前和后续的位都不会受影响。后面要讨论的，这不是自同步流密码的方式。

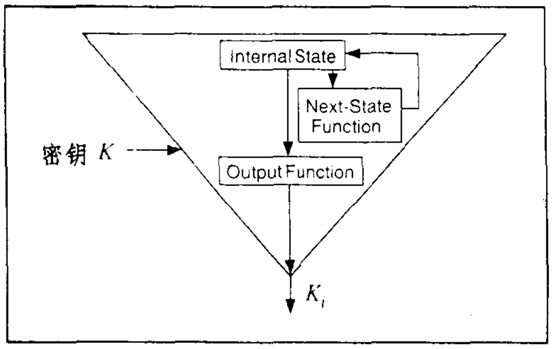


图 8.13 密钥流发生器内部结构

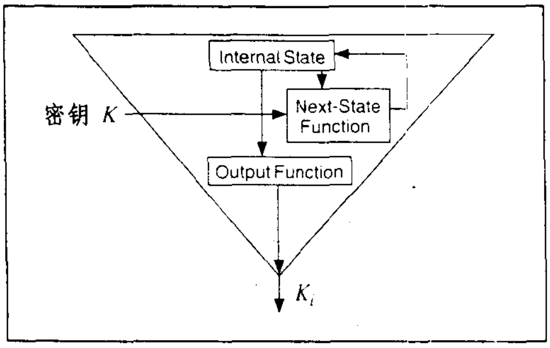


图 8.14 输出反馈方式的密钥流发生器

同步流密码也能防止任何密文中的插入和删除，因为它们会引起同步丢失并立即检测出来。然而，它们不能完全防止位固定。类似于 CFB 方式的块密码，攻击者能固定流中的个别位，如果他知道明文，他可将这些位解密成任何报文，后续位的解密是完全扭曲的（直到系统恢复），但在一定的场合攻击者还是能做破坏。

按照内部操作方式，同步流密码也可进一步分为输出反馈方式和计数器方式。

输出反馈方式

输出反馈方式的流密码中，密钥影响下一状态函数（如图 8.14），输出函数不依赖于密钥：往往简单到就像内部状态的某一位，或者内部状态多位的异或。其密码复杂性在于下一状态函数：该函数依赖于密钥。这种方法也称为内部反馈，因为反馈机制在密钥生成算法的内部。

这种方式也有变形形式：密钥 K 只决定密钥流发生器的内部状态，密钥设置了内部状态以后，发生器从此“安详”地运行。

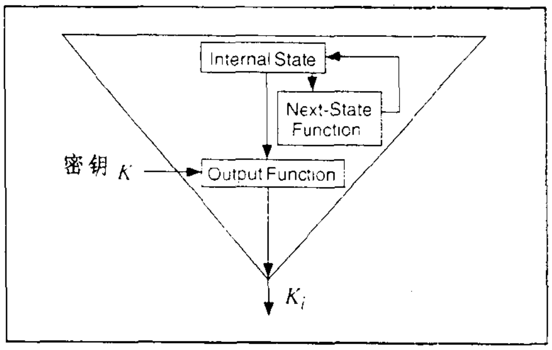


图 8.15 计数器方式的密钥流发生器

计数器方式

计数器方式的流密码中，下一状态函数比较简单，输出函数比较复杂并依赖于密钥，如图 8.15 所示。下一状态函数简单到就像是一个计数器，在前一状态中递增。计数器方式流密码的

密码复杂性在于输出函数。

插入攻击

输出反馈方式和计数器方式对插入攻击都是敏感的。攻击者已经记录了一个密文流但不知道明文或用于加密该明文的密钥流。

原明文： $p_1 p_2 p_3 p_4 \cdots$

原密钥流： $k_1 k_2 k_3 k_4 \cdots$

原密文： $c_1 c_2 c_3 c_4 \cdots$

攻击者把已知的一位 p' 插入到明文中 p_1 的后面，并设法得到以同一密钥流加密的修改后的明文，他记录下产生的新密文：

新明文： $p_1 p' p_2 p_3 p_4 \cdots$

原密钥流： $k_1 k_2 k_3 k_4 k_5 \cdots$

新密文： $c_1 c'_2 c'_3 c'_4 c'_5 \cdots$

假设他知道 p' 的内容，他可以从原密文和新密文中确定 p' 后面的整个明文：

$$k_2 = c'_2 \oplus p' \qquad 则\ p_2 = c_3 \oplus k_2$$

$$k_3 = c'_3 \oplus p' \qquad 则\ p_3 = c_3 \oplus k_3$$

$$k_4 = c'_4 \oplus p' \qquad 则\ p_4 = c_4 \oplus k_4$$

攻击者甚至不需知道 p' 插入的具体位置：他只要将原密文和新密文对比较一下，看看从哪里开始不同。为防止这种攻击，千万不要对二个不同报文采用同一个密钥流加密。

8.3.2 自同步流密码

自同步流密码中，密钥流的每一位是前面固定数量的密文位的函数，自同步流密码通常以密码反馈方式操作。

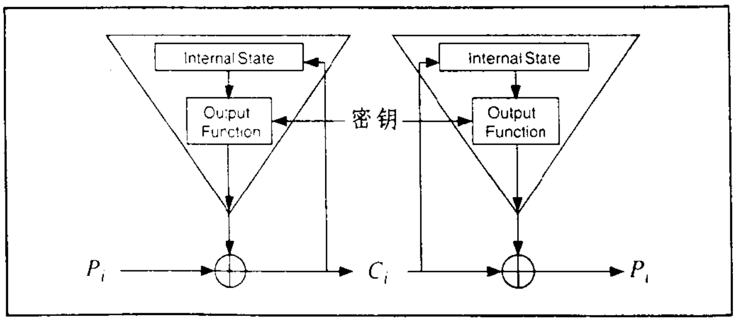


图 8.16　密码反馈方式的密钥流发生器

图 8.16 是密码反馈方式的流密码，内部状态是前面 n 位密文的函数，其密码复杂性在于输出函数，输出函数根据内部状态产生一位密钥流。

由于内部状态完全取决于前面 n 位密文，在接收到 n 位密文以后，解密的密钥流发生器自动与加密的密钥流发生器同步。

这种方式的一个实现技巧是每个报文加上 n 位随机的报文头，报文头加密、传输、再解密，解密结果是不正确的，但是在该 n 位之后二个密钥流发生器就可同步。

自同步流密码的缺点是错误扩散。对传输过程中篡改了的每位密文，解密的密钥流产生器会输出 n 位错误的密钥流，所以密文中的每个错误会引起 n 个相应的明文错误，直到篡改了的

位离开内部状态。

自同步流密码对重发也是敏感的。攻击者记录了一些密文位，后来他把记录下来的密文块替换当前正在传输的密文，接收端在接收到一些杂乱的信息以后重新同步，原密文将正确地解密出来。接收端无法知道这不是当前的数据，但原数据得到重发。除非使用了时戳，攻击者通过重发同样的报文就可以一次又一次地让银行在他的帐号上存款（当然假设密钥没有更换）。

8.3.3 块密码作为流密码使用

块算法可以以任何方式用作密钥流发生器，块算法为密码系统提供密码安全性。

图 8.17 是一个输出反馈方式的流密码，其中块算法充当输出函数的角色。内部状态的最低位成为密钥流发生器的输出，密钥流发生器的内部状态成了块算法的输入，块算法用于修改内部状态，可以有一种或二种修改方式。块算法的整个输出作为新的内部状态，或者输出中的某一位用于以某种简单的方式修改内部状态，有时内部状态仅仅是来自下一状态函数的前 n 位。

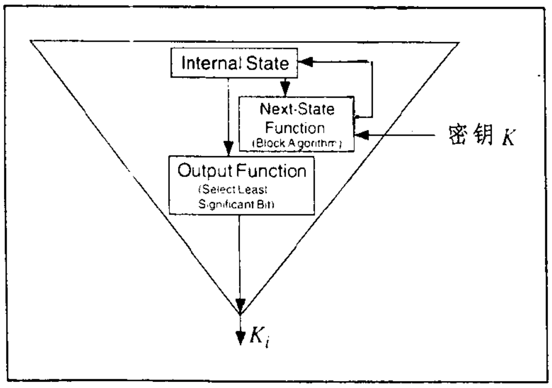


图 8.17 采用块算法的输出反馈方式的密钥流发生器

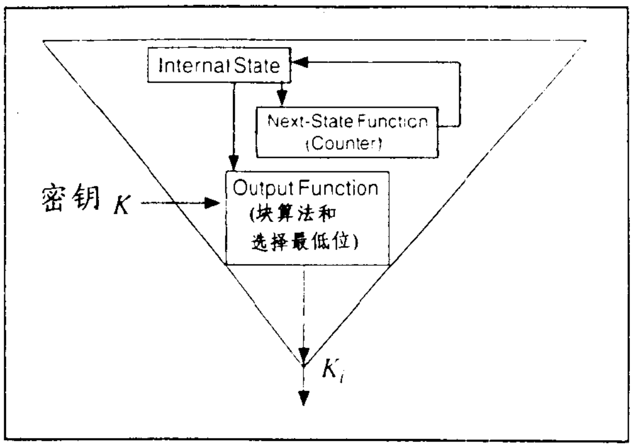


图 8.18 采用块算法的计数器方式的密钥流发生器

图 8.18 是一个计数器方式的流密码，其中块算法起输出函数的作用。整个内部状态是块算法的输入，块算法输出的最低位是密钥流发生器的输出。下一状态函数只是一个计数器。

图 8.19 是一个密文反馈方式的流密码，其中块算法作为输出函数。整个内部状态是块算法的输入，块算法输出的最低位是密钥流发生器的输出。内部状态为前面的 n 位密文。

8.4 流密码和块密码

考虑流密码和块密码之间的差异在实用上比理论上更重要。流密码逐位加密解密而不真正适合于软件实现，由于位操作比较慢，块算法更适合于软件实现。

从密码术的角度来看，块密码更为普遍，原则上块密码的算法更为坚固一些。另一方面，流密码易于从数学角度来分析。两者之间的本质区别在于错误扩散，块密码中，一个错误至少篡改了一块数据，这在各种反馈方式的块密码中更为严重。流密码可以做到篡改一位密文只会篡改一位明文。有的场合为发现错误需要有足够的错误扩散，而有些场合却绝不能有错误扩散。

另外也得考虑加密数据的类型。如果要从计算机终端上加密 ASCII 数据，64 位的块算法不见得非常合适，而需要一个能加密 7 或 8 位 ASCII 字符的算法。另一方面，计算机磁盘中的数据是整块读写的，这对块算法是很理想的。

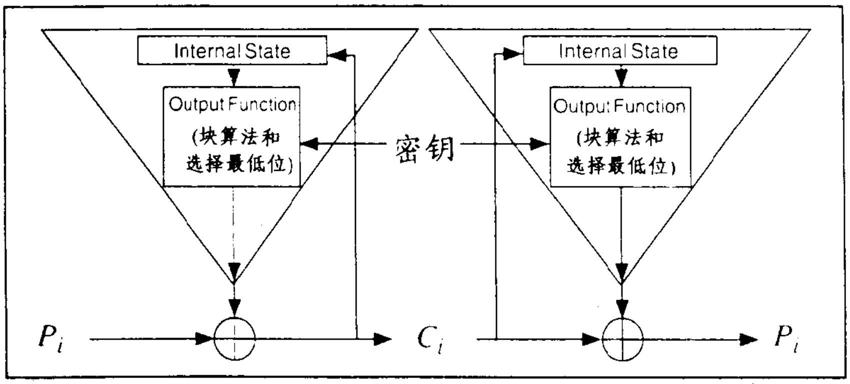


图 8.19　采用块算法的密文反馈方式的密钥流发生器

8.5　公钥密码术和对称密码术

公钥密码术和对称密码术哪个更好？自从公钥密码术问世以来就一直进行着这场争论，然而，这场争论是假设两者之间可以公平比较的。争论哪个更安全是没有意义的，任何算法的安全性依赖于钥长，大多数对称算法都有具体的钥长，公钥算法有不定的钥长。以固定的 56 位密钥的 DES 要比以 40 位密钥的 RSA 安全得多，以 1000 位密钥的 RSA 又要比 DES 安全得多。

Needham 和 Schroeder 指出以公钥协议的报文的数量和长度要比以对称协议的大得多，他们的结论是对称协议比公钥协议有效得多。虽然这是正确的，但这个分析忽略了公钥方法的优秀的安全性。

W. Diffie 写道：

“把公钥密码术看作为一种新的密码系统的形式，而不是一种新的密钥管理的形式时，我不考虑安全性和性能。敌手很快指出 RSA 系统的速度大约是 DES 的千分之一，而所需的钥长为 DES 的十倍。虽然，使用公钥系统可以减少常规密码术所需的密钥交换，这一点从一开始就很明显，但对其必要性却并不是立即明白的。在这种背景下，建立一种混合的系统的提议，像是一个大发现而被欢呼。”

公钥密码术和对称密码术能很好地协同工作。公钥密码术强在密钥分发和协议，公钥密码术可以完成对称密码术永远完成不了的功能。另一方面，对称密码术的速度比公钥密码术快好几个数量级。对称算法适用于加密文件和通讯信道，公钥算法适合于加密和分发密钥、提供认证和完成所有前面各章描述的功能。

8.6　加密通讯网络

理论上，可以在开放系统互连(OSI)参考模型的任何一层施行加密操作，不过加密执行点不同，有不同的问题需要考虑。

链路加密

链路加密是在物理层执行加密(如图 8.20)。物理层的接口是最标准的，易于连接硬件加密设备。加密设备加密所有经过的数据，包括数据、路由信息、协议信息等，可用于任何类型的

数字通讯链路。另一方面，发送者和接收者之间的所有智能转换和存储节点都需要将数据流解密之后才能处理。

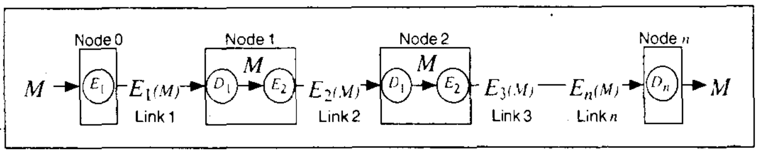


图 8.20 链路加密

这种加密方式是很有效的。由于所有数据都加密了，密码分析者无法得到信息的结构，他们无法知道交换双方是谁？发送的信息多长？什么时间通讯等等。这称为通讯流安全性(traffic-flow security)：敌手不仅不能取得信息，而且也无法了解信息发送到何处和信息有多长。

系统安全性只依赖于选择的算法，而与通讯管理技术无关。密钥管理也是很简单的，只有链路的二端需要一个共同的密钥，他们可以不考虑网络的其它部分而独立更换他们的密钥。

设想一个以 1 位 CFB 加密的同步通讯线路。初始化以后，线路就不停地工作，自动从位错误和同步错误中恢复。当有报文要发送时，线路加密信息，否则它仅加解密随机数据。攻击者无法知道通讯何时开始、何时结束，她看到的只是无穷无尽的看似随机的位流。

如果该通讯线路是异步的，同样可以采用 1 位 CFB 方式，其差异在于敌手可以得到有关传送率的信息。如果必须隐藏传送率信息，需要采取一些措施在线路空闲时发送一些假的报文。

在物理层加密的最大问题是网络中的所有物理链路都得加密，任何没有加密的链路都会危及整个网络的安全性。如果网络比较大，这种加密方式所需的费用是高得惊人的。

另外，由于网络中每个节点都处理没有加密的数据，所以每个节点都得保护，除非网络中所有的用户都相互信赖，并且所有节点都很安全，但这是不可能的。即使在某个公司内，信息也得在部门内保密。如果网络偶然将信息送错地方，人人都可以读到。

端对端加密

另一种方法是将加密设备放在网络层和传输层之间。加密设备要能够按照协议理解低三层的数据，并且只加密传输层数据，传输层数据与没有加密的路由信息组合在一起发送给低层传输。

这种方法避免了物理层加密的问题。通过端对端加密(end-to-end encryption)，数据直到到达最终目的地都保持加密的形式(如图 8.21)。端对端加密的主要问题是数据的路由信息没有加密，这样一个好的密码分析者无需知道对话内容，而从下列信息中就可获悉许多详情：交谈双方是谁？何时交谈？交谈时间多长？由于每个用户都得确信他们相互之间有共同的密钥，密钥管理也困难得多。

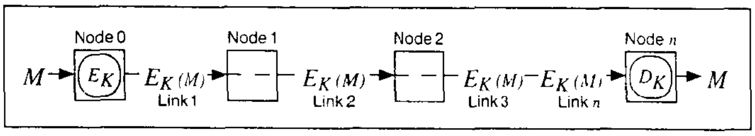


图 8.21 端对端加密

如果在通讯体系结构的最高层加密，如在应用层或表示层，这将与通讯网络使用的类型无关。这还是端对端加密，但加密实现不需要考虑线路编码、Modem 之间的同步、物理接口等。早期电机密码术的时候，加密和解密都完全离线(off-line)进行，与此也只是一步之遥。

在这些层的加密操作会与相应的软件相互影响。对于不同的计算机体系结构，这些软件也是不同的，所以为适用于不同的计算机系统，必须优化加密操作。加密可以软件自身或专用硬件来实现。在后一种情况下，计算机先把数据发送给专用硬件加密，再发送给通讯体系结构的低层传输。这些操作需要一些智能终端而不适合于非智能终端。另外，不同类型的计算机之间可能还会有兼容性问题。

混合加密

把端对端加密和链路加密组合在一起是保护网络的最有效方法。每个物理链路的加密使不可能对路由信息作任何分析，而端对端加密减少了网络中各个节点上没有加密的数据的威胁。这二种方案的密钥管理可以完全分开，网络管理人员管理物理层上的加密，而各个用户负责端对端加密。

8.7 加密数据存储

为存储和日后检索而加密数据不同于为通讯而加密数据，具有完全不同的问题：

1. 密码分析者攻破算法所需的时间仅受限于他们认为这些数据在多长时间内有价值。

2. 数据也可以以明文形式存放在其它磁盘、计算机或纸上，这给密码分析者执行已知明文攻击提供了更多的机会。

3. 在数据库应用中，数据片可能比大多数算法的块小得多，这会引起密文比明文大得多。

4. I/O 设备的速度要求快速的加密和解密，可能需要硬件加密，在某些应用场合也许需要特殊的高速算法。

5. 需要安全、长期地存储密钥。

6. 由于不同的人要访问不同的文件或者同一文件的不同部分等，密钥管理是非常困难的。

如果加密文件不是象记录或域(field)这样有结构的，例如文本文件，那么检索是容易的，整个文件在使用之前解密。如果解密文件是数据库文件，那也存在较大问题，为访问单条记录而将整个数据库解密是很低效的，但是单独解密记录也许对块重发之类的攻击是敏感的。

另外，必须确保在加密之后删除未加密文件。

8.8 硬件加密和软件加密

硬件

直到最近，所有加密产品都是以专用硬件的形式出现，它们是一些插入到通讯线路中并对经过的所有数据加密的一种加密/解密盒。虽然今天软件加密比以前流行，硬件仍然不失为大多数商业和军事应用的选择方式，这其中有许多原因。

首先是速度问题。象下一章所要讨论的，加密算法含有许多对位串的复杂操作，这不是普通计算机内部的那种操作。最普遍的二个加密算法，DES 和 RSA，含有微处理器上许多低效的操作步骤。虽然许多密码编制人员致力于使算法更适合于软件实现，安全性始终是首要的设计动力。专用硬件始终在速度上占优势。

另外，加密往往是一项高计算量的任务，为此占用计算机的微处理器是低效的。把加密的任务移向另一个芯片，即使该芯片是另一个处理器，将加快整个系统的速度。

硬件加密流行的第二个原因是安全性问题。加密算法以软件实现没有物理保护。攻击者可以利用各种调试工具私下修改算法，而任何人都不会发觉。硬件加密设备可以安全地封装以防止这种情况。防篡改的盒子可以防止有人修改硬件加密设备，专用 VLSI 芯片可以套上一种

化学物，这样任何试图访问芯片内部都会破坏芯片的逻辑。美国政府的 Clipper 和 Capstone 芯片都设计成防篡改的。芯片可以设计成攻击者不可能读到未加密的密钥。

IBM 在大型机上为加密数据和通讯开发了一个密码系统，它含有一个防篡改的模块用以维护密钥。

电磁辐射有时可以重现一块电子设备内部的执行内容。专用的加密盒可以防护电磁辐射，这样就不会泄露信息。虽然通用计算机也同样可以做到防护，但这是一个相当复杂的问题。美国军方称之为 TEMPESE，这问题远远超出了本书的范围。

硬件加密流行的最后一个原因是易于安装。大多数加密场合没有通用计算机，人们希望加密他们的电话会谈、传真机或数据链路，在电话机、传真机和 Modem 上安装专用加密硬件要比安装在微处理器或软件上有效得多。

即使加密的数据来自于计算机，安装专用硬件加密设备还是要比修改计算机系统软件容易。加密对用户应该是透明的，软件上要做到对用户透明的唯一方法是把加密操作编程深入到操作系统内部，这是不容易的。另一方面，即使计算机初学者也能在计算机和 Modem 之间安装加密盒。

当前市场上有三类基本的加密硬件：自身含有加密功能的模块（执行诸如口令校验和密钥管理等功能），对通讯链路的专用加密盒，以及 PC 插卡。

有的加密盒是为某种类型通讯链路设计的。例如，T-1 加密盒不能用于加密同步位，优化传真加密器用于这类数据。有的加密盒是用于同步通讯线路，而其它有的盒子用于异步通讯线路。其趋势是向更改的位率和更大的使用灵活性发展。

虽然如此，其中许多设备有硬件或软件的不兼容性，购买时应注意并弄清具体需求，以免加密设备不能完成任务，注意硬件类型、操作系统、应用软件、网络等等的限制。

PC 插卡通常将写入硬盘的所有内容加密，并可以选择配置成加密写入软盘和串行口的所有内容。PC 插卡不能防护电磁辐射和物理干扰，因为如果计算机不受影响的话，保护这些插卡也并没有什么益处。另外，更多的公司开始把加密硬件装入通讯设备上，安全的电话机、传真机和 Modem 都有成品。

这些设备的内部密钥管理有许多不同的方法，一般都是安全的。有的方法只适合于一种场合，购买时应该知道加密盒中采用哪类密钥管理和他们希望提供什么功能。

软件

所有的加密算法都可以软件实现。其缺点在于速度、费用和易于使用，优点在于灵活性和轻便性。

所有主要的操作系统都有大量的软件加密程序，这些程序是指定保护个别文件的，用户一般需要手工加密和解密某个文件。保证密钥管理方法安全是很重要的，不把密钥存放在磁盘上（或者甚至写在存储器中由处理器交换到磁盘的某处），解密之后要删除密钥和未加密的文件。

8.9 文件删除

当在计算机上删除文件时，文件并非真正删除了，唯一删除了的只是磁盘索引文件的条目。许多软件商通过销售恢复已删除文件的文件恢复软件而发了财。

为删除文件使得这些软件包不能将其恢复，需要将磁盘上所有的文件位物理重写。按照美国国家计算机安全中心(NCSC)的说法：

“重写是在原来存储敏感数据的位置写入非保密数据的过程。…为清除…存储介质，DoD

首先以某模式重写，再以它的补，最后以另一模式重写。例如首先重写 00110101，再写入 11001010，最后写入 10010111。所需重写的次数取决于存储介质，有时取决于数据的敏感性，有时取决于不同 DoD 需求。总之，直到最后用非保密数据重写才算完成文件清除。”

大多数商业程序都声称实现 DoD 标准重写三次，第一次全 1，第二次全 0，最后以重复的 1-0 模式重写。笔者推荐对删除的文件重写七次，第一次全 1，第二次全 0，和五次以密码安全的伪随机序列重写。美国国家标准和技术局(NIST)最近因电子隧道显微镜取得的进展，提出即使如此重写也许还是不够。

还有一个担忧是虚拟存储器的使用意味着计算机可以随时把存储器内容读或写到磁盘上，即使用户没有存盘，用户不会知道计算机什么时候将正在处理的敏感文档写入磁盘，所以偶尔把磁盘上所有没有使用的空间和交换文件以及文件的最后一块的剩余部分也重写一下是必要的。

8.10 选择算法

当要评价算法并为个人使用选择算法时，人们有许多选择：

1. 可以选择出版发表的算法，基于相信发表的算法已经过许多密码编制人员的详细审阅。如果没人能够攻破，它肯定相当不错。

2. 可以信赖产商，基于相信熟知的产商有良好的声誉，不太可能冒险销售劣等算法的设备而影响其声誉。

3. 可以信赖私人顾问，基于相信公正的顾问对市场上不同算法作可靠评价是训练有素的。

4. 可以信赖政府，基于相信政府是值得信赖的，不会误导其公民。

5. 可以自己设计算法，基于相信自己的密码才能是一流的，并且不信赖任何人。

当然每一种选择都会有问题，但第一种选择的问题少一些，只信赖某个产商、顾问或政府可能是自找麻烦，大多数自称为安全顾问的人(甚至是著名公司的成员)往往对加密一无所知，大多数安全性的产商也好不了多少。美国国家安全机构(NSA)有许多世界一流的密码编制人员，但很难知道是否他们会和盘托出他们的知识。除非你是个天才，自己设计的算法未经对等检查就使用简直是十足的傻瓜。

本书中所有的算法都是公开的，它们已公开出版发表并已经过该领域专家的分析，这些算法可能是比较安全的。

第三部分 密码算法

第9章 数学基础

9.1 信 息 论

申农于 1948 年创立了现代信息论，他的论文集最近由 IEEE 出版社再版，本节仅概述一些主要的论题，有关信息论完善的数学讨论请参见有关文献。

熵和不确定性

信息论中定义一个信息的信息量为对整个信息进行编码所需的最少位数。例如数据库中“星期”字段所用长度不超过 3 位信息，因为该信息可以用 3 位编码：

000 ---- Sunday

001 ---- Monday

010 ---- Tuesday

011 ---- Wednesday

100 ---- Thursday

101 ---- Friday

110 ---- Saturday

111 未用

如果该信息用相应的 ASCII 字符串表示，将占用更多的存储空间但无更多的信息。类似地，“性别”字段即使可以用 6 字节 ASCII 字符串"MALE"或"FEMALE"来存储，但只有 1 位信息量：1 表示男、0 表示女或反之。

在正式场合，信息 M 中的信息量用信息的熵(entropy)来度量，记作 H(M)。信息的熵表示“性别”为 1 位，“星期”为略小于 3 位。信息的熵一般为 $\log_2 n$，其中 n 是所有含义的数量，并假设每一个含义是同等可能的。

信息的熵同样度量了信息的不确定性(uncertainty)，它也说明了把密文恢复为明文时的可能性。例如，如果密文块"QHP * 5M"不是"MALE"就是"FEMALE"，则信息的不确定性为 1。密码分析者只需懂得恰当选取 1 位信息量就可恢复密文。

如果明文的不确定性太小，公钥密码系统对仅知密文攻击是脆弱的。例如，如果 C＝E(P)，这里 P 是 n 个可能明文之一，则密码分析者只需将所有的 n 个可能明文加密并把结果与 C 比较(加密密钥是公开的)。这种方法他们不能找到加密密钥，但能决定已知密文对应的明文 M(M 是 n 个可能明文中的一个)。

例如，如果 P 是小于 100 万的款额，这种攻击方法就是密码分析者尝试所有可能的款额(概率加密解决这个问题)。对称密码系统对这类攻击是不脆弱的，因为密码分析者没有密钥就不能尝试加密操作。

语言的速率

对一种给定的语言，语言的速率(rate of language)为

$$r = H(M)/N$$

其中N为信息的长度。正规英语由于N很大,其速率在1.0—1.5位/字母之间(本书取申农的估计值1.2)。语言的绝对速率(absolute rate)为每个字符序列同等可能的情况下编码每一个字符所需的最大位数。如果语言中有L个字符,绝对速率为:

$$R = \log_2 L$$

这是单个字符的最大熵。

英语有26个字母,绝对速率为$\log_2 26 = 4.7$位/字母。应该不会惊奇英语的实际速率比绝对速率小得多,因为英语是种高冗余的语言。

语言的冗余度(redundancy),记作D,定义为

$$D = R - r$$

假设英语速率为1.2,冗余度为3.5位/字母,这意味着每个英语字符具有1.2位信息,其余都是冗余的。以英语字母构成的ASCII文本中每8位文本还是只有1.2位信息,这意味着它有3.5位冗余信息,所以每位ASCII文本中大约有0.56位信息的冗余度。以EBCDIC表示同一文本,每个字符有5位信息,每位EDCDIC文本有0.30位冗余度和0.70位的熵。空格、标点符号、数字和格式化信息要修改这个结果。

密码系统的安全性

申农为密码系统何为安全定义了一个精确的数学模型。密码分析者的目标是确定密钥k、明文p或者二者都要确定,然而,他们可能满足于有关p的一些概率性信息,如是否是数字化的音频信息、是否是德文、是否是图表数据等等。

在大多数真正的密码分析中,分析者在着手分析之前拥有有关p的一些概率性信息。他们可能知道加密的语言,这种语言具有某一个冗余度。如果是送给Bob的报文,很可能是以"Dear Bob"开头。密码分析的目的是通过分析来修改与每一个可能明文相关的概率,最终总归会有一个明文从一大堆可能的明文中出现(或者至少,非常可能)。

密码系统要达到理想保密,即密码系统中的密文不会产生有关明文的任何可能信息(除其长度之外)。申农在理论上指出唯一的可能是密钥数至少与报文数一样多,就是说,密钥必须至少与报文自身一样长,并且密钥不能再次使用,即一次一密。

暂且不谈理想保密,密文不可避免地会产生一些有关相应明文的信息,好的密码算法把这种信息减少到最小,优秀的密码编制人员利用这些信息来确定明文。

密码破译人员利用语言的固有冗余度来减少可能的明文数。语言的冗余度越大,越易于密码破译。这就是为什么许多真正的密码实现中在加密之前用压缩工具缩短文本的大小。压缩也减少报文的冗余度。

密码系统的熵是密钥空间大小的尺度,近似为以2为底的密钥数的对数:

$$H(K) = \log_2(\text{密钥数})$$

以64位密钥的密码系统的熵为64,56位密钥的密码系统的熵为56。一般而言,熵越大,密码系统越难以攻破。

唯一解距离

对于一个长度为n的报文,为把密文解密成与原明文同一种语言的有意义的明文,不同的密钥数由下式给出:

$$2^{H(K)-nD} - 1$$

申农定义唯一解距离(unicity distance),也称为唯一解点(unicity point)为当相应明文中

的熵与加密密钥的熵之和等于密文的位数时所需的密文量的近似值，他进一步说明当密文比该距离长时肯定只有一种有意义的解密结果，当密文比该距离短时可能会有多个同等有效的解密结果，由此从敌手选择正确的解密结果的难度中获取了安全性。

对于大多数对称密码系统，唯一解距离定义为密码系统的熵除以语言冗余度的商：

$$U = h(K)/D$$

如同所有的统计或信息论的度量，唯一解距离不做确定的预测，只给出概率结果。唯一解距离表示当执行蛮力攻击时，可能只有唯一有意义的明文解密结果所需的最少密文量。一般而言，唯一解距离越大，密码系统越好。对于以 56 位密钥的 DES 以及 ASCII 英语报文，唯一解距离为：

$$56/6.8 = 8.2$$

大约 8.2 个 ASCII 字符，或 66 位。

唯一解距离不是度量密码分析需要多少密文，而是度量密码分析为得到只有一个合理解而所需多少密文。即使理论上以少量密文就可能攻破的密码系统，可能计算上是不可破的。唯一解距离与冗余度成反比，当冗余度接近 0 时，即使一段微小的密文也是不可破的。

信息论的实践

虽然上述概念具有很好的理论价值，实际的密码分析很少沿着这条路线处理。如果唯一解距离太小则必定是不安全的，但大的唯一解距离并不保证安全性。很少有实际的算法是绝对分析不出的，所有的特征都可用于攻破加密报文，然而，类似的信息论考虑有时也是有用的，例如，用于确定某算法的密钥更换间隔。密码分析者也采用各种统计和信息论测试来帮助在最有希望的方向上引导分析。

不幸，有关信息统计在密码分析中的应用的大量文献，包括 1940 年图灵的文章，还都是针对经典密码学的。

混乱和扩散

按照申农的观点，在明文中遮掩冗余度的二个基本技术是混乱(confusion)和扩散(diffusion)。

混乱技术遮掩了明文和密文之间的关系，这就挫败了试图从寻找冗余度和统计特性来研究密文。最简单的实现方法是替代(substitution)。简单的替代密码，如凯撒替代，是把明文中每一个相同的字母替代成密文的同一个字母。现代替代密码更复杂一些，把明文的一长块替代成密文的不同块，并且替代时也改变了明文的每一位。这类替代未必是充分的，例如德国的 Enigma 是个复杂的替代算法，但在计算机面前被攻破了。

扩散技术通过把明文扩展到整个密文中来消散明文的冗余度。密码分析者要化费大量的时间才能发现这些冗余度。实现扩散的最简单方法是置换(transposition)，也称为排列(permutation)。简单的置换密码，如列置换，只是简单地把明文字母重新排列。现代密码也是做这种置换，但也采用其它扩散方式把部分报文扩散到整个密文报文中。流密码只采用混乱技术，块算法既采用了混乱技术又采用了扩散技术，只采用扩散技术是易于攻破的。

9.2 复杂性理论

复杂性理论为分析不同密码技术和算法的计算复杂性(computational complexity)提供了方法论，它把各种密码算法和技术作比较并确定其安全性。信息论告诉我们所有的密码算法(除一次一密之外)都是可攻破的，复杂性理论告诉我们是否在有限的时间内可攻破它们。

算法的复杂性

密码强度由攻破它所需的计算量决定，算法的计算复杂性由二个变量度量：T(时间复杂性)和S(空间复杂性)，T和S一般都表示成n的函数，其中n为输入量。本节中，引用一个“破译算法”用于攻破密码。

算法的计算复杂性一般用“大O”记号表示其数量级，复杂性的数量级只是当n增大时增长得最快的复杂性函数的项，而忽略所有常量和低次的项。例如，某算法的时间复杂性为$4n^2+7n+12$，则计算复杂性在n^2数量级，记作$O(n^2)$。

时间复杂性的这种度量方法的优点是独立于系统，无需知道各种指令的精确执行时间、用于表示不同变量的位数以及处理器的速度。计算机的速度、数据传输速度各有差异，但算法复杂性的数量级仍然相同。

这种表示方法可以方便地描述输入量对时间和空间需求的影响。例如，如果$T=O(n)$，输入量加倍将使算法的运行时间加倍。如果$T=O(n^2)$，则输入量中增加1位会加倍算法的运行时间。

算法通常按照时间和空间复杂性分类。复杂性独立于n，即O(1)的算法是常量级的，复杂性以n线性增长，即O(n)的算法是线性级的，同样也有平方级、立方级等的算法。所有这些算法是多项式级的，其复杂性为$O(n^t)$，其中t为常量。具有多项式时间复杂性的算法称为多项式时间算法。

复杂性为$O(t^{f(n)})$的算法称为指数级的算法，其中t为常量，f(n)为n的多项式函数。复杂性为$O(t^{f(n)})$，其中t为常量，f(n)为比常量数量级大但比线性级小的函数，这类算法称为超多项式级的算法。

密码编制人员理想中希望所有的破译算法都是指数级时间复杂性的，而实际上，当前计算复杂性理论指出：“所有已知破译算法都是超多项式级的时间复杂性的”，就是说，对于实际中使用的密码，我们已知的破译算法是超多项式级的时间复杂性的，但还不能证明不存在多项式级时间复杂性的破译算法。计算复杂性理论的进展也许会有一天设计出一个密码，这种密码以数学确定性理论排除了多项式时间复杂性的破译算法的存在。

当n增大时，算法的时间复杂性之间有巨大的差异。表9.1说明各类算法的运行时间，其n为10^6。

表9.1　各类算法的运行时间

类别	复杂性	$n=10^6$时需要的操作次数	每秒执行10^6次操作时的算法运行时间
常量级	O(1)	1	1μs
线性级	O(n)	10^6	1s
平方级	$O(n^2)$	10^{12}	11.6天
立方级	$O(n^3)$	10^{18}	32年
指数级	$O(2^n)$	$10^{301.030}$	宇宙年龄的10^{301}倍

考虑一下对密码系统作蛮力攻击的问题。攻击的时间复杂性与可能的密钥数，即钥长的指数成比例，如果n为钥长，蛮力攻击的复杂性为$O(2^n)$。对56位密钥的DES蛮力攻击的时间复杂性为2^{56}(2285年，假设每秒测试10^6次)，对112位密钥的复杂性为2^{112}(10^{20}年，基于同样假设)，前者已经似乎不可能了，后者是绝对不可能的。

问题的复杂性

复杂性理论也根据求解问题所需的算法来分类问题。复杂性理论在理论计算机——图灵

机(Turing machine)上考虑求解问题中最难的例程的最小时间和空间需求。图灵机是具有无限读/写存储器的一个有限状态机,已经证明图灵机是计算的理想模型。

可以用多项式时间算法解决的问题称为可处理的(tractable),因为这些问题通常可以对合理的输入量在合理的时间内求解(“合理”的精确定义依赖于环境)。不能在多项式时间内求解的问题称为不可处理的,因为计算会迅速变得不可行,不可处理的问题有时就称之为难问题。即使 n 值相对较小,以指数或超多项式级的算法可以求解的问题也是计算不可处理的。

图灵证明了有的问题是非确定的,不可能设计出任何算法来求解,更不用说用多项式时间算法。

问题可按复杂性分成各类,每一类依赖于求解的复杂性。图 9.1 是更重要的复杂性类及其它们之间假设的关系(遗憾的是有关这些材料并没有多少已经在数学上得以证明)。

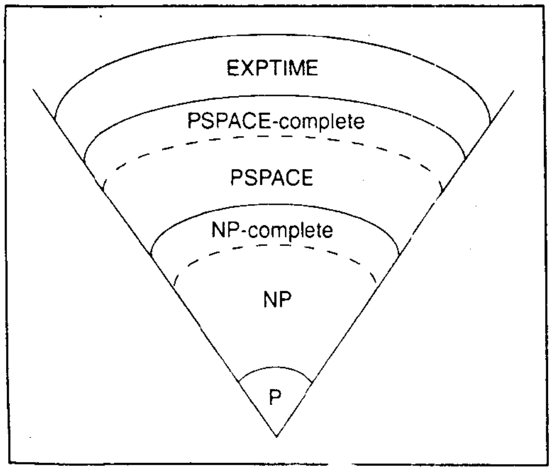


图 9.1 复杂性类

最底层的 P 类是由所有在多项式时间内可解的问题组成,NP 类由所有在非确定图灵机上能在多项式时间内求解的问题组成。非确定图灵机是一般图灵机的变型,它能够做猜测,这种机器猜测问题解答并在多项式时间内检测猜测结果。

NP 与密码术的关系如下:许多密码的常规类可在非确定的多项式时间内攻破。给出密文 C,密码分析者简单地猜测一个明文 X 和密钥 K,并且在多项式时间内以 X 和 K 作输入运行加密算法并检验结果是否为 C。这在理论上是很重要的,因为它为这类密码的密码分析复杂性置了一个上界,实际上密码分析者寻找的当然是一个确定的多项式时间算法,而且,这个论点并不能适用于所有的密文类,事实上,它不能适用于一次一密——对任何的 C,当运行加密算法时有许多 X 和 K 对都能产生 C,但这些 X 中大多数都是没有意义的,不是合理的明文。

NP 类包含了 P 类,因为任何在确定图灵机上能在多项式时间内求解的问题必定在非确定图灵机上能在多项式时间内求解,只需简单地略去猜测阶段即可。

如果在确定机上在多项式时间内所有 NP 问题都可求解,则 P=NP。很显然有的 NP 问题比其它要难得多(如对密文蛮力攻击和将明文的随机一块加密),但至今还未证明 P 是否等于 NP 的问题。然而,所有从事复杂性理论的学者都猜想它们是不等的。

NP 类中有些特殊问题可以证明是与该类中所有问题同样难。Cook 证明了“满足性问题”(给定一个命题的逻辑公式,是否存在一种对各个变量真假值的配置方法,使得公式为真)是 NP 完全问题。这意味着,如果在多项式时间内满足性问题可解,那么 P=NP。反之,如果可以证明任何一个 NP 问题没有确定的多项式时间算法,那么满足性问题也不会有确定的多项式时间算法。这样,满足性问题是最难的 NP 问题。自从 Cook 的创举性论文发表以来,人们发现了与满足性问题等价的大量问题。等价地,与任何 NP 问题同样难的问题也是 NP 完全问题。如果它们能在确定的多项式时间内求解,那么 P? NP 问题也就解决了。P? NP 是计算复杂性理论中没有解决的主要问题,如果证明了 P=NP,那么许多类密码都可以在非确定的多项式时间内轻易破译,并且也可由确定算法破译。

再外层等级的是 PSPACE 类，PSPACE 问题可在多项式空间内求解，但不是必定在多项式时间内。PSPACE 包含 NP，但 PSPACE 中有的问题比 NP 更难，当然这也没有得到证明。有一类称为 PSPACE 完全问题，具有如下性质：如果其中任何一个问题在 NP 内，则 PSPACE=NP，并且如果其中任何一个问题在 P 内，则 PSPACE=P。

最后是一类称为 EXPTIME 的问题，这些问题可在指数级时间内求解。EXPTIME 完全问题实际上可以证明在确定多项式时间内是不可解的，就是说，已经证明 P 不等于 EXPTIME。

NP 完全问题

M. Garey 和 D. Johnson 编制了 300 多个 NP 完全问题，这里列出其中的几个：

旅行售货商问题。一位旅行售货商要到 n 个不同的城市，是否存在一条路径使他经过每个城市一次且仅一次(这是 Hamiltonian 回路的推广)？

三方婚姻问题。现有 n 位男人、n 位女人和 n 位牧师，同时有一张可接受的婚姻组合列表，每个组合为一位男人、一位女人和一位主持婚礼的牧师。在这种三人组合列表的情况下，是否可能有一种婚姻安排，使 3n 位人中的每位都结婚或主持婚礼？

三元满足性问题。现有 n 个逻辑命题的列表，每个命题有 3 个逻辑变量：if (x and y) then z, (x and w) or (not z), if ((not u and not x) or (z and (u or not x))) then ((not z and u) or x)等等。是否存在一种对各个变量真假值的配置方法，使得所有命题均为真(这是满足性问题的一种特例)？

9.3 数　论

本节介绍一些对密码术有重要意义的数论要点，除非明确指出，只考虑整数算术。

模算术

模算术也称为"时钟算术"。某人说他 10 点钟回家，但他迟到了 13 个小时，他是几点钟回家的？这是模 12 算术。

$$(10+13) \bmod 12 = 11$$

另一种写法是

$$10+13 = 11 \pmod{12}$$

一般如果 $a=b+kn$，其中 k 为常数，则 $a=b \pmod n$。如果 a 和 b 都是正数并且 $a<n$，则 a 即为 b 除 n 时的余数。一般 a 和 b 除 n 时的余数相同。有时称 b 为 a 模 n 的余数，有时称 a 与 b 模 n 同余。

从 0 到 $n-1$ 的整数集构成称为模 n 的完全余数集，就是说，对于任何整数 a，它模 n 的余数在 0 到 $n-1$ 之间。

注意这里的 mod 定义与有的程序设计语言中的定义不同。例如，PASCAL 中的模运算有时返回一个负数，它的返回值在 $-n-1$ 到 $n-1$ 之间。C 中的%运算符返回第一个操作数除以第二个操作数的余数，如果二者之一为负数，则返回一个负数。本书中的所有算法，如果返回一个负数时要在结果上加以 n。

模算术类似于普通算术：它是可交换、可结合和可分配的。化简每一个模 n 中间结果与整个运算求模再化简模 n 结果是一样的。

$$(a+b) \bmod n = ((a \bmod n)+(b \bmod n)) \bmod n$$
$$(a-b) \bmod n = ((a \bmod n)-(b \bmod n)) \bmod n$$
$$(a * b) \bmod n = ((a \bmod n) * (b \bmod n)) \bmod n$$

$$(a*(b+c)) \bmod n = (((a*b) \bmod n)+((a*c) \bmod n)) \bmod n$$

密码术中大量使用模 n 计算，因为诸如计算离散对数和平方根是比较困难的，而模运算限制了所有中间值和结果的范围。对于一个 k 位的模数 n，任何加法、减法和乘法的中间结果不会超过 2k 位，所以可以用模算术作指数运算，而不产生大的中间结果。

计算 a 的幂次方的模运算：

$$a^x \bmod n$$

只是一连串的乘法和除法。有加快运算速度的方法，因为运算是可分配的，以一连串连续的乘法，每次求模的方法做指数运算会快得多。尤其对大数(如 200 位长)运算时其优点更为明显。

例如计算 $a^8 \bmod n$，不用常规的方法执行 7 次乘法和一次大的模化简：

$$(a*a*a*a*a*a*a*a) \bmod n$$

而执行 3 次小的乘法和 3 次小的模化简：

$$((a^2 \bmod n)^2 \bmod n)^2 \bmod n$$

同样

$$a^{16} \bmod n = (((a^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n$$

计算 $a^x \bmod n$，当 x 不是 2 的幂次方时稍稍难一点。n 的二进制表示是一些 2 的幂次方之和，例如 25 的二进制为 11001，即 $25=2^4+2^3+2^0$，故

$$\begin{aligned} a^{25} \bmod n &= (a * a^8 * a^{16}) \bmod n \\ &= (a * ((a^2)^2)^2 * (((a^2)^2)^2)^2) \bmod n \\ &= ((((a^2 * a)^2)^2)^2 * a) \bmod n \end{aligned}$$

合理地存储中间结果，只需 6 次乘法：

$$((((((a^2 \bmod n) * a) \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n) * a) \bmod n$$

C 语言实现该算法为：

```
int modexp(int a, int x, int n)
{
    int i=1;
    while (x>0)
      {
      if (x%2)     /* is x odd ? */
      r=(r*a)%n;
      a=(a<<1)%n;
      a<<=1;
      }
  return r;
}
```

如果 k 为数据的比特数，这种技术平均只需 1.5k 次操作，找到最少操作的方法是比较困难的(已经证明至少需要 k－1 次操作)，但当 k 增大时，把操作次数减少到 1.1k 次或更少一些并非很难。

指数模 n 的逆操作是计算离散对数，这将在后面讨论。

素数

素数是指比 1 大、只有 1 和自身二个因子的数，即没有别的数能除尽它。例如 2，73，2521，

2365347734339，$2^{756839}-1$ 以及 1996 年 11 月发现的 $2^{1398269}-1$ 都是素数。素数有无穷多个，密码术往往使用大素数（如 512 比特长甚至更大）。

E. Kranakis 写了一本有关素数及其在密码术中的应用的优秀著作，P. Ribenboim 写了二篇有关素数的优秀论文。

最大公因子

当二个数 a 和 n 除 1 之外没有其他公因子时称它们互素（relatively prime），即 a 和 n 的最大公因子是 1，记作

$$\gcd(a,n)=1$$

15 和 28 互素，15 和 27 不是，13 和 500 也互素。一个素数与除它的倍数之外的所有数都互素。

求二个数的最大公因子的最简单方法是用欧几里德算法（Euclid's algorithm），欧几里德在"Elements"一书中描述该算法（作于大约公元前 300 年），该算法并非欧几里德发明的，历史学家相信在公元前 500 年左右就已有了该算法。这是存活到至今还保持完好的最古老的不平常算法。Knuth 描述了该算法并作了一些修改。

C 语言描述欧几里德算法为：

```
/* returns GCD of x and y, assuming x and y are >0 */
int gcd(int x, int y)
{
    int g;
    if (x<0) x=-x;
    if (y<0) y=-y;
    g=y;
    while (x>0)
    {
    g=x;
    x=y%x;
    y=g;
    }
  return g;
}
```

该算法可推广到返回 m 个数的最大公因子：

```
/* returns the GCD of x1,x2...xm, assuming all x are >0 */
int multiple_GCD(int m, int *x)
{
    int i,g;
    if (m<1) return 0;
    g=x[0];
    for (i=1; i<m; i++)
    {
    g=gcd(g,x[i]);
```

```
    /* optimization, since for random x[i],
      g==1 60% of the time : */
    if (g==1) return 1;
    }
    return g;
}
```

模 n 的逆元

在普通乘法中，4 的逆元为 1/4，因为 4 * 1/4 = 1。在模运算中，求逆问题会复杂得多：

$$4 * x=1 \bmod 7$$

这个方程等价于求解 x 和 k，使得

$$4x=7k+1$$

这里 x 和 k 都为整数。

一般问题是寻找一个 x，使得

$$1=(a * x) \bmod n$$

也可写成

$$a^{-1}=x \pmod n$$

模求逆元问题有时有解，有时没有。例如对于模 14，5 的逆元为 3，因为 5 * 3=15=1(mod 14)，而 2 没有逆元。

如果 a 和 n 互素，$a^{-1}=x \pmod n$ 一般有唯一解，如果 a 和 n 不是互素，方程无解。如果 n 为素数，则 1 到 n－1 中的每一个数都与 n 互素，并在这个范围内恰好有一个逆元。

那么，如何求模 n 的逆元？方法有好几种，欧几里德算法也可计算一个数模 n 的逆元，有时称之为推广的欧几里德算法。

C 语言描述如下：

```
static void Update(int *un, int *vn, int q)
{
    int tn;
    tn= *un- *vn *q;
    *un= *vn;
    *vn=tn;
}

/* return == gcd(x,n) == u*u1+v*u2 */
int Extended_Euclidian(int u, int v, int *u1_out, int *u2_out)
{
    int u1=1, u3=u, v1=0, v3=v, q;
    while (v3>0)
    {
    q=u3/v3;
    Update(&u1,&v1,q);
    Update(&u3,&v3,q);
```

```
    }
    *u1_out=u1;
    *u2_out=(u3-u1*u)/v;
    return u3;
}
```

该算法是迭代计算的，当数据比较大时速度比较慢。Knuth 指出这个算法平均除法次数为

$$0.843 * \ln(n) + 1.47$$

求解系数

欧几里德算法可用于求解下面问题：给定 m 个变量 $x_1, x_2, \cdots, x_m$，求解 m 个系数 $u_1, u_2, \cdots, u_m$，使得

$$u_1 * x_1 + \cdots + u_m * x_m = 1$$

费马(Fetmat)定理

如果 m 为奇数，a 不是 m 的倍数，则

$$a^{m-1} = 1 \pmod n$$

欧拉(Euler) Totient 函数

另有一种计算模 n 逆元方法，但并不总是可以采用。模 n 的余数缩减集是余数完全集中与 n 互素的元素构成的子集，例如，模 12 的余数缩减集是{1,5,7,11}。如果 n 为素数，模 n 的余数缩减集为{x|1<=x<=n-1}，0 永远不是余数缩减集的元素。

欧拉 Totient 函数，也称为欧拉 FAI(n)函数，是模 n 的余数缩减集中的元素个数，即 FAI(n)是小于 n 且与 n 互素的正整数个数。

如果 n 为素数，则 FAI(n)=n-1。如果 n=p*q，这里 p 和 q 均为素数，则 FAI(n)=(p-1)*(q-1)。这些数字在后面讨论的公钥算法中经常出现。

按照费马定理的欧拉推广，如果 gcd(a,n)=1，则

$$a^{FAI(n)} \bmod n = 1$$

计算模 n 的 a-1 为：

$$x = a^{FAI(n)-1} \bmod n$$

例如，对于模 7，求解 5 的逆元。由于 7 为素数，FAI(7)=7-1=6，故 5 的逆元为

$$5^{6-1} \bmod 7 = 5^5 \bmod 7 = 3$$

求逆的二个算法都可推广在一般问题

$$(a * x) \bmod n = b$$

中求解 x(如果 gcd(a,n)=1)。

用欧拉推广：

$$x = (b * \exp(a^{FAI(n)-1} \bmod n)) \bmod n$$

用欧几里德算法：

$$x = (b \times inv(a,n)) \bmod n$$

如果 gcd(a,n)不为 1，则(a * x) mod n = b 也能有多解或无解。一般而言，欧几里德算法求逆要快一些，尤其对于 500 比特以内的数据。

中国余数定理

在素数 n 的因子分解基础上，可以用中国余数定理求解整个方程组，这个定理在公元一世

纪由孙子发现。

一般地，如果素数因子分解为 $n=p_1*p_2*\cdots*p_t$，则方程组：

$$(x \bmod p_i) = a_i \qquad \text{其中 } i=1,2,\cdots,t$$

有唯一解 x，这里 x 小于 n。（注意这里有的素数可以重复出现，如 p_1 可以等于 p_2）。

这样，对于任意 $a<p$ 和 $b<q$（p 和 q 均为素数），存在唯一的 x，这里 $x<p*q$，使得

$$x = a \pmod p \text{ 且 } x = b \pmod q$$

为求解 x，首先用欧几里德算法求解 u，使得

$$u * q = 1 \pmod p$$

再计算

$$x = (((a-b)*u) \bmod p)*q+b$$

中国余数定理的 C 语言实现为：

```
/* r is the number of elements in arrays m and u;
    m is the array of (pairwise relatively prime) moduli
    u is the array of coefficients
    return value is n such that n==u[k]%m[k]
        k=0..r-1 and n<m[0]*m[1]*... *m[r-1]      */
int Chinese_Remainder(int r, int *m, int *u)
{
    int i,modulus,n;
    modulus=1;
    for (i=0; i<r; i++)
    modulus *=m[i];
    n=0;
    for (i=0; i<r; i++)
    {
    n+=u[i]*modexp(modulus/m[i], totient(m[i]),m[i]);
    n%=modulus;
    }
    return n;
}
```

中国余数定理的一个推论可用于求解类似的问题：如果 p 和 q 为素数，且 $p<q$，则存在唯一的 x，这里 $x<p*q$，使得

$$a = x \pmod p \qquad \text{且} \qquad b = x \pmod q$$

如果 $a>=b \bmod p$，则

$$a = (((a-(b \bmod p)*u) \bmod p)*q+b$$

如果 $a<b \bmod p$，则

$$a = (((a+p-(b \bmod p)*u) \bmod p)*q+b$$

平方余数

如果 p 为素数且 $a<p$，对于一些 x，有

$$x^2 = a \pmod p$$

则 a 为模 p 的平方余数(quadratic residue)。

并非所有的 a 都满足这个性质，例如，如果 p=7，平方余数为 1，2 和 4：

$$1^2 = 1 = 1 \pmod 7$$

$$2^2 = 4 = 4 \pmod 7$$

$$3^2 = 9 = 2 \pmod 7$$

$$4^2 = 16 = 2 \pmod 7$$

$$5^2 = 25 = 4 \pmod 7$$

$$6^2 = 36 = 1 \pmod 7$$

注意这里每个平方余数出现二次。

没有任何 x 满足下列方程：

$$x^2 = 3 \pmod 7$$

$$x^2 = 5 \pmod 7$$

$$x^2 = 6 \pmod 7$$

模 7 的平方非余数(quadratic nonresidue)为 3，5 和 6。

容易证明恰好有(p－1)/2 个模 p 的平方余数和等量个平方非余数。同样，如果 a 为模 p 的平方余数，则 a 正好有二个平方根：一个在 0 到(p－1)/2 之间，另一个在(p－1)/2 到(p－1)之间，其中一个平方根也是模 p 的平方余数，这称为平方根原则。

如果 n=p * q，这里 p 和 q 均为素数，则正好有(p－1)(q－1)/4 个模 n 的平方余数，一个模 n 的平方余数正好是模 n 的平方。例如，模 35 有 6 个平方余数：1，4，9，11，16，29，每个平方余数恰好有 4 个“平方根”。

Legendre 符号

当 a 是任何整数并且 p 为大于 2 的素数时，定义 Legendre 符号为：

L(a,p) = 0　如果 a 除尽 p

L(a,p) = 1　如果 a 为模 p 的平方余数

L(a,p) = －1　如果 a 为模 p 的平方非余数

计算 L(a,p)的简便方法为：

$L(a,p) = a^{(p-1)}/ 2 \bmod p$

雅可比(Jacobi)符号

雅可比符号，记作 J(a,n)，是 Legendre 符号的推广，它定义在任何整数对 a 和 n 上，这个函数揭示测试是否为素数。雅可比函数作用于 n 的约数的缩减集上，可由下列公式计算：

J(a,n) = 1　如果 n 为素数且 a 为模 n 的平方余数

J(a,n) = －1　如果 n 为素数且 a 为模 n 的平方非余数

J(a,n) = J(h,p1) * ... * J(h,pm)　如果 n 为合数

　　　　这里 $p_1 \ldots p_m$ 为 n 的素数因子分解

计算雅可比符号的递归算法为：

1. J(1,k)=1

2. J(a * b,k)=J(a,k) * J(b,k)

3. 如果(k²－1)/8 为偶数，则 J(2,k)=1，否则 J(2,k)=－1

4. J(b,a)=J((b mod a),a)

5. 如果 GCD(a,b)=1，则：

a. 如果$(a-1)(b-1)/4$为偶数，则$J(a,b) * J(b,a)=1$

b. 如果$(a-1)(b-1)/4$为奇数，则$J(a,b) * J(b,a)=-1$

算法的C语言实现为：

```
int jacobi(int a, int b)
{
    int a1,a2;
    if (a>=b) a%=b;
    if (! a || a==1) return a;
    if (a==2)
    if (! ((b*b-1)/8)%2) return 1;
    else return -1;
    if (a & b & 1) /* both a and b are odd */
        if (! ((a-1)*(b-1)/4)%2) return jacobi(b,a);
        else return -jacobi(b,a);
    if (gcd(a,b)==1)
        if (! ((a-1)*(b-1)/4)%2) return jacobi(b,a);
        else return -jacobi(b,a);
    factor2(a,&a1,&a2);
    return jacobi(a1,b)*jacobi(a2,b);
}
```

如果p为素数，计算雅可比符号的更好方法是：

$J(a,p) = 1$　　若$a=1$

$J(a,p) = J(a/2,p) * (-1)^{(p^2-1)/8}$　　若a为偶数

$J(a,p) = J(p \bmod a,a) * (-1)^{(a-1)*(p-1)/4}$　　若a为奇数且a不为1

注意这也是判断a是否为模p的平方余数的有效方法(要求p为素数)，如果$J(a,p)=1$，则a为平方余数，如果$J(a,p)=-1$，则a不是余数。

同时注意如果$J(a,n)=1$但n为合数，则a未必是模n的平方余数。例如

$$J(7,143) = J(7,11) * J(7,13) = (-1)(-1) = 1$$

然而没有一个整数x满足$x^2=7 \bmod 143$。

要使a成为模n的平方余数，a必须是模所有n的素数因子的平方余数。

Blum 整数

如果p和q均为素数，且都与3模4同余，则$n=p*q$是Blum整数。如果n为Blum整数，每个平方余数恰好有4个平方根，其中之一也是平方余数，这是平方根原则。例如，139模437的平方根原则为24，其它三个平方根为185,252和413。

Blum整数用于有的公钥算法中。

生成元

如果p为素数，且$g<p$，如果对每一个从0到$p-1$的n，存在a使得

$$g^a = n \pmod p$$

则g为模p的生成元，或称作g为p的本原元(primitive)。

例如，如果$p=11$，2为模11的生成元：

$$2^{10}=1024=1 \pmod{11}$$
$$2^{1}=2=2 \pmod{11}$$
$$2^{8}=256=3 \pmod{11}$$
$$2^{2}=4=4 \pmod{11}$$
$$2^{4}=16=5 \pmod{11}$$
$$2^{9}=512=6 \pmod{11}$$
$$2^{7}=128=7 \pmod{11}$$
$$2^{3}=8=8 \pmod{11}$$
$$2^{6}=64=9 \pmod{11}$$
$$2^{5}=32=10 \pmod{11}$$

从 1 到 10 中的每个数都可表示成 $2^a \pmod p$。

对于 p=11，生成元为 2,6,7 和 8，其它都不是生成元，例如，3 不是生成元因为下式无解：

$$3^a = 2 \pmod{11}$$

一般地，找到一个生成元也并非易事。然而，如果 p－1 可素数因子分解成 $p-1=q_1 * q_2 * \ldots * q_n$，要判断 g 是否为模 n 的生成元，只需对所有的 $q=q_1, q_2, \ldots, q_n$，计算

$$g^{(p-1)/q} \pmod p$$

如果对有的 q 值上式为 1，则 g 不是生成元。如果对所有的 q 值上式均不等于 q，则 g 为生成元。

例如，设 p=11，p－1=10 的素数因子分解为 2 和 5，测试 2 是否为生成元：

$$2^{(11-1)/5} \pmod{11} = 4$$
$$2^{(11-1)/2} \pmod{11} = 10$$

没有一个结果为 1，故 2 为生成元。

测试 3 是否为生成元：

$$3^{(11-1)/5} \pmod{11} = 9$$
$$3^{(11-1)/2} \pmod{11} = 1$$

所以，3 不是生成元。

有限域内计算

如果 n 为素数，则我们有了有限域(finite field)(其实如果 n 为素数的幂次方时也是有限域)，所以把 n 换成 p。有限域也称为伽罗瓦(Galois)域，记作 GF(p)。

在有限域内也定义了加法、减法、乘法和除法运算。加法有一个幺元 0，乘法有一个幺元 1，每一个非 0 元素有唯一的逆元(如果 p 不是素数则非然)。交换律、结合律和分配律也都成立。

有限域内的算术大量用于密码术中，所有的数论在有限域内都适用，只是每个数的大小有限，并且除法不会有任何舍入错误。许多密码系统都基于 GF(p)，其中 p 为大素数。

在 GF(2^n)内计算

密码编制人员使用算术模 n 次不可约多项式，多项式的系数为模 q 的整数，q 为素数。这种域称为 GF(q^n)。所有的算术都模 p(x)运算，其中 p(x)为 n 次不可约多项式。

例如 GF(2^3)有下列元素：$0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1$。也有在 GF($2^n$)内求逆的算法，并且适合于并行实现。

提及多项式，术语“素数”往往替代成“不可约(irreducible)”，多项式不可约是指它不能表示成二个其它多项式之积(当然除 1 和自身之外)。多项式 x^2+1 在整数上是不可约的，而 x^3+

$2x^2+x$ 不是，它可表示成 $x(x+1)(x+1)$。

在给定域内是生成元的多项式称为本原元(primitive)。

用线性反馈移位寄存器可快速硬件实现在 $GF(2^n)$ 内的计算，因为如此，在 $GF(2^n)$ 上计算往往比在 $GF(p)$ 上运算快，就象在 $GF(2^n)$ 上指数运算更为高效而用于计算离散对数。

对于有限域 $GF(2^n)$，密码编制人员往往采用三项式 $p(x)=x^n+x+1$ 作为模，因为 x^n 和 x 之间的一长串 0 系数使得易于实现快速的模数乘法。三项式必须是本原元。构成 x^n+x+1 为本原元的小于 1000 的 n 值有：

1，3，4，6，9，15，22，28，30，46，60，63，127，153，

172，303，471，532，865，900

已有实现 $GF(2^{127})$，其中 $p(x)=x^{127}+x+1$ 的硬件问世，并有文章讨论了在 $GF(2^n)$ 内实现指数运算的硬件体系结构。

9.4 因子分解

对于一个数因子分解是指求解其素数因子：

10 = 2 * 5

60 = 2 * 2 * 3 * 5

252601 = 41 * 61 * 101

$2^{113}-1$ = 3391 * 23279 * 65993 * 1868569 * 1066818132868207

因子分解问题是数论中最古老的困难之一，检测一个数是否为素数是简单的，只需测试小于该数的所有素数中是否有它的因子。现代算法不需测试所有的素数，但还是对可能的素数逐个测试，不过已经取得了长足的改进。

目前最好的因子分解算法为：

平方筛(quadratic sieve)算法。这是对 150 十进制位以下的数因子分解已知最快的算法，并广为采用。该算法的一个更快的形式称为多个多项式平方筛(multiple polynomial quadratic sieve)算法，最快的形式称为多个多项式平方筛的双倍大素数形式(double large prime variation of multiple polynomial quadratic sieve)。

数域筛(number field sieve，NFS)算法。这是已知最快的因子分解算法，本算法最先提出时不能使用，但由于近年来一系列的改进已经可以使用了。对于小的数据(平均在 110 到 135 十进制位左右，依赖于实现方法)还是多个多项式平方筛算法快一些，NFS 用于因子分解第九费马数。

还有其它因子分解算法，但大部分情况都由上述二种算法取代，如椭圆曲线方法(Elliptic Curve Method，ECM)，Pollard 蒙特卡洛算法(Pollard's Monte Carlo Algorithm)，连分数算法(Continued Fraction Algorithm)，试探除法(trival division)等。

如果 n 为欲因子分解的数，最快的平方筛方法执行的操作次数基本为：

$$e^{(\ln n)^{\frac{1}{2}}(\ln(\ln n))^{\frac{1}{2}}}$$

对于 664 比特(200 位)的数，结果约为 10^{23} 次操作。以每秒执行 100 万次操作(一次操作为算法的一次复杂迭代，而不是指微机的操作)的速度也需 37 亿年。

对于 120 位以上的数，数域筛方法快一些，渐近的启发式时间估计为：

$$e^{(\ln n)^{\frac{1}{3}}(\ln(\ln n))^{\frac{2}{3}}}$$

对于许多特殊形式的数，如 $2^{523}-1$，NFS 要快得多。

1970 年发布的一条重要的新闻是因子分解了一个 41 位的难数，“难数”是指没有任何小的因子，也不易于分解的特殊形式的数。十年之后，由 Cray 大型机因子分解二倍大小的难数也只需几个小时，如今，110 位的难数也可以规则地因子分解，这主要是由于因子分解理论和计算机硬件的发展。

C. Pomerance 采用定制的 VLSI 芯片设计了一台模块式因子分解机，能做因子分解的数的大小取决于机器的大小及价格。他的样机价值 25,000 美元，可在 2 周内因子分解 100 位的数，价值 1000 美元的机器可在 1 年内因子分解 150 位的数。理论上，这种因子分解机没有上限，在 1 年内因子分解 200 位的数的机器价值 1000 亿美元。

除了大规模的并行机之外，在计算机网络上成功地实现因子分解算法已经取得重大进展。Lenstra 和 Manase 利用一些全球计算机的空闲时间在几个月内因子分解了一个 116 位的数，相当于一台 400 MIPS 的计算机运行一年，所有机器之间用 E－Mail 通讯。

这种操作方法只能得到改进。Lenstra 和 Manase 的实现方法可以扩展到具有更多机器的网络上。由于计算机性能的每提高十倍可以使因子分解的数加长稍微超过 10 位，所以由一些大学或大公司的工作站网络在几周或几个月的空闲时间内因子分解 130 位的数据是不太可能的，而且这些工作站要比 Pomerance 的因子分解机普遍得多。Lenstra 估计在 50 台大机器上用 NFS 算法因子分解一个 155 位(512 比特)的数需要 1 年，并且他认为该估计是保守的。

因子分解是个快速发展的领域，如果没有开发出新的方法，2048 比特的数(由二个 1024 比特的素数相乘而得)做因子分解是不太可能的，然而，没人能够预测未来，在数域筛算法问世之前，许多人都推测平方筛算法与其它因子分解方法的速度差不多，然而他们错了。

模 N 的平方根

如果 n 为二个素数的乘积，则计算模 n 的平方根的可能性计算上等价于将 n 因子分解的可能性，换言之，知道了 n 的素数因子可以容易地计算一个数模 n 的平方根，但是已经证明计算模 n 的平方根与计算 n 的素数因子同样难。

9.5 素数生成

把二个大素数相乘被推测为是一个单向函数，即把二个数相乘是容易的，而把乘积因子分解来恢复这二个大素数是很难的。

测试一个整数是否为素数则是完全另一回事，并且要比因子分解容易得多。

公钥密码术需用素数，任何合理地排列的网络需用大量的素数，可以有多种方法生成这些素数。

首先解决一些明显的问题：

1. 如果人人需要一个不同的素数，素数会用完吗？不会。其实，512 比特以内的素数不止 10^{150} 个。对于大小为 N 的数，一个随机数为素数的可能性大约为 logN。宇宙中只有 10^{84} 个原子，如果每个原子从问世到现在每微秒需要十亿个素数，则还剩下 10^{110} 个素数。

2. 如果二个人偶然选择同一个素数会如何？二个人从 10^{150} 个素数中选择同一个素数的可能性比计算机自燃的可能性还要小，这无需担忧。

因子分解一个 200 位的数需要这么长的时间，找到一个 200 位的素数会容易多少？如果用先产生一个随机数，再试着对其因子分解的方法寻找素数也许容易不了多少。然而，有多种测试方法可以确定一个随机数是否为素数，其可能性在 50%(或者更高)以上。如果一个数没有

通过测试，它肯定不是素数。如果一个数通过了一项测试，至多50%可能性不是素数。如果通过了二项测试，至多25%可能性不是素数。如果一个数通过了十项测试，它不是素数的可能性至多1/1024。

如果测试100次整个测试过程失败的可能性为$1/2^{100}$，约为$1/10^{30}$。如果需要更加确定一个数是素数，可以选择更多的随机数再测试一下。

Solovay-Strassen 法

Solovay 和 Strassen 设计了一个素数可能性的测试算法，这个算法用了 Jacobi 函数。

为测试 p 是否为素数：

(1)选择一个小于 p 的随机数 a

(2)如果 gcd(a,p)不为1，则 p 测试失败，p 为合数

(3)计算 $j=a^{(p-1)/2} \bmod p$

(4)计算 Jacobi 符号 J(a,p)

(5)如果 j 不等于 J(a,p)，则 p 肯定不是素数

(6)如果 j=J(a,p)，则 p 不是素数的可能性最多50%

用 n 个不同的随机数 a 重复测试 n 次，则通过所有 n 次测试时 p 为合数的可能性最多为$1/2^n$。

该算法的实现可以快速执行，生成一个大的随机数 p 以测试是否为素数，检验保证不能被小素数2,3,5,7,11等整除，再生成另一个随机数 r，并计算 gcd(p,r)和 $r^{(p-1)/2} \bmod p$，执行各种测试，如果通过了则生成另一个随机数 r 重新测试直到满意为止。

Rabin-Miller 法

这个算法有 Rabin 部分基于 Miller 的思想而设计的，其实，这是 NIST 的 DSS 提议中的算法的简化形式。

首先选择一个随机数 p 以测试是否为素数，计算 b(其中 2^b 为除以 p－1 的最大的2的幂次方)，再计算 m，使得 $n=1+2^b m$。

(1)选择一个小于 p 的随机数 a

(2)置 j=0，$z=a^m \bmod p$

(3)如果 z=1 或 z=p－1，则 p 通过了测试，可能为素数

(4)如果 j>0 且 z=1，则 p 不是素数

(5)置 j=j+1。如果 j<b 且 z 不等于 p－1，置 $z=z^2 \bmod p$ 回到(4)

(6) 如果 j=b 且 z 不等于 p－1，则 p 不是素数

这个测试方法比前者收敛得更快一些，它不超过$1/4^n$时间(n 为迭代次数)就可判断一个数不是素数。对于一个不能被小素数整除的数执行12次(或无论多少次)测试，结果几乎必定是素数。

Lehmann 法

另一个更简单的测试方法是由 Lehmann 独立设计的，设置迭代次数为100的算法如下：

(1)选择一个随机数 n 以测试是否为素数

(2)保证 n 不能被小素数整除，测试2,3,5,7,11会明显加快算法

(3)在1和 n－1之间选择100个随机数 $a_1, a_2, ..., a_{100}$

(4)对所有 $a_i=a_1, a_2, ..., a_{100}$，计算 $a_i^{(n-1)/2} \pmod n$，如果某个 a_i 测试失败则停止

(5)如果对所有 i，$a_i^{(n-1)/2}=1 \pmod n$，则 n 可能为合数。如果对任何 i，$a_i^{(n-1)/2}$不等于1

或－1 (mod n)，则 n 为合数。如果对任何 i，$a_i^{(n-1)/2}=1$ 或－1(mod n)，但不是对所有的 i 都为 1，则 n 为素数。

强素数

许多 RSA 文献都建议对 p 和 q 应采用强素数(strong prime)，强素数是指具有某些性质的素数，使得其乘积 n 难以用特定的因子分解方法来分解，已经建议的性质有：

p－1 和 q－1 的最大公因子要小

p－1 和 q－1 都有大的素数因子，分别为 p′和 q′

p′－1 和 q′－1 都有大的素数因子，分别为 p″和 q″

(p－1)/2 和(q－1)/2 都是素数

是否需要强素数是个争论的问题，这些性质是用于阻碍一些因子分解算法，然而，这些性质并不妨碍最快的因子分解算法作因子分解。

然而这种情况可能会改变，也许会设计出新的因子分解算法，对具有某些性质的数作因子分解反而会更方便，如果这样，也许会再次需要强素数。

即使强素数对加大因子分解的难度是不必要的，还是最好采用强素数，虽然生成素数时更难一些，但用了也无妨。

9.6　有限域中的离散对数

模指数运算是常用于密码术中的另一个单向函数，计算下列表达式是容易的：

$a^x \bmod n$

模指数运算的逆问题是求一个数的离散对数，这是一个难问题：

求 $a^x=b \bmod n$ 中的 x

例如

如果 $3^x \bmod 17 = 15$，则 x＝6

并非所有的离散对数都是有解的(有效解只能是整数)，显然下面方程中的 x 是无解的：

$3^x \bmod 13 = 7$

求解 1024 比特的数的这类问题更加难。

有限组内计算离散对数

下面三个主要的有限组中的离散对数是密码编制人员关心的：

素数域的倍乘数：GF(p)

有限域中的幂次组：$GF(2^n)$

有限域 F 上的椭圆曲线组：EC(F)

许多公钥算法的安全性都是基于求离散对数的问题，所以大家都广泛地在研究该问题。文献中可以了解到该问题的综合概况和当时的最佳解决方法。

如果 p 为模且为素数，则在 GF(p)上求离散对数的复杂性基本上等同于对同样大小的整数 n 作因子分解，这里 n 是二个基本上等长的素数之积，复杂性为：

$$e^{(\ln p)^{\frac{1}{2}}(\ln(\ln p))^{\frac{1}{2}}}$$

NFS 算法快一些，渐近的启发式时间估计为

$$e^{(\ln p)^{\frac{1}{3}}(\ln(\ln p))^{\frac{2}{3}}}$$

不过还是不实用。

如果 p 为模，p－1 至少有一个大因子，则计算离散对数会困难得多，否则，在 GF(p)内计

算离散对数会容易得多。

Pohlog 和 Hellman 设计了一个如果 p－1 只有小素数因子时在 GF(p) 内计算离散对数的快速方法，正因为如此，密码术没有采用有的域。另外一个算法可以与因子分解的速度相当的计算离散对数：扩展到 $GF(p^n)$ 形式的域，这个算法理论上存在问题。

当前在素数域内计算离散对数有三种方法：线性筛(linear sieve)、Guassian 整数法和 NFS。

主要的需要执行的计算量每个域中只有一次，在此之后，可以快速计算各个算法，这对基于这些域的系统来说是个安全性的弱点。不同的用户采用不同的素数域是很重要的，幸运的是，有许多素数，故有许多素数域。

在多项式域中，研究者没有忽略 $GF(2^n)$。Coppersmith 算法可在 $GF(2^n)$ 之类的域中求离散对数并在 $GF(2^{400})$ 域中求解也勉强可行，算法的预计算阶段是巨大的，但该算法的确很有效。该算法的一种稍微低效的实现中，在 7 个小时预计算之后，每几秒钟就可在 $GF(2^{127})$ 内求出离散对数。

第 10 章　数据加密标准

10.1　数据加密标准

数据加密标准(Data Encryption Standard, DES)自从 1977 年 1 月 5 日颁布以来已经成为世界标准,除非面遇强手有破译的可能,十多年来 DES 对密码破译保持良好的安全性。

背景和历史

70 年代初,非军事的密码研究是杂乱无章的,该领域内几乎没有发表研究论文。大多数人都知道军事部门采用特殊的编码设备通讯,但很少有人懂得密码科学。美国国家安全机构(NSA)有大量的知识,但他们不能公布其内容。

美国一些小公司生产和销售密码设备,主要是销往他国政府,这些设备都是相互不同的,设备之间不能通讯并且没人确切知道这些设备是否安全,没有独立的机构对这些设备验证其安全性,购买这些密码设备的用户也不知道他们购买的究竟是何物。

1972 年,美国国家标准局(NBS),即现在的国家标准和技术局(NIST)开始了一项保护计算机和通讯数据的计划,其中一项内容是研制一个标准密码算法来保护在传输和存储着的数字化数据。单一算法可以测试和验证,可以在不同的密码设备上相互操作,这种算法也应易于实现和获得。

NBS 在 1973 年 5 月 15 日的"联邦登记册(Federal Register)"上发布了一则向社会公开征集标准密码算法的消息,他们指定了一系列设计准则:

- 算法必须提供高度的安全性
- 算法必须描述完整并易于理解
- 算法的安全性在于密钥,即安全性不依赖于算法的保密
- 算法对所有用户都有用
- 算法必须能适用于不同应用场合
- 用电子设备实现必须经济
- 必须使用起来高效率
- 必须能验证
- 必须是可出口的

公众的反响表示出对密码标准极有兴趣,由于该领域内的专家非常少,一些数学家送来了算法的粗框,但没有一个设计满足要求。

NBS 在 1974 年 8 月 27 日再次发布了征集告示,最后收到了令人满意的设想它是基于 IBM 在 70 年代初开发的一个算法,称为 LUCIFER,该算法虽然较复杂但很明确,它只用相对较少的数值上的简单逻辑运算,可以用硬件和软件相当有效地实现。

NBS 请求 NSA 帮助评价该算法的安全性并确定是否适合于作为联邦标准。IBM 已经取得了该算法的专利,但愿意把该智力财产转让给他人生产、实现和使用,NBS 和 IBM 协商生产和销售实现该算法的器件的合理价格。

最后，NBS 在 1975 年 3 月 17 日同时公布了算法的细节和 IBM 允许算法技术转让的声明，另外在 1975 年 8 月 1 日发布征集公开的研究和讨论的告示。

当时许多密码编制人员怀疑 NSA 修改了算法并设置了“陷门”(trap door，设计者暗置其中的机关以达简易解密的目的)，他们抱怨 NSA 把钥长从原来 Lucifer 的 112 比特减少到 56 比特，他们抱怨算法的内部操作，这些在 70 年代似乎神秘和令人烦恼的问题，则在 90 年代初才弄清楚它的原因。

1976 年 NBS 举办了二次研讨会来评价该提议标准，第一次研讨会讨论算法的数学问题和陷门的可能性，第二次讨论增加算法钥长的可能性。研讨会邀请了算法设计者、评价者、实现者、销售者、使用者和评论家，从所有报导来看，讨论会非常活跃。

尽管如此，数据加密标准在 1976 年 11 月 23 日采纳为联邦标准并被授权用于所有非保密的政府通讯。该标准的官方说明书，FIPS PUB 46，“数据加密标准”在 1977 年 1 月 15 日发行；FIPS PUB 81，“DES 操作方式”于 1980 年发行；FIPS PUB 74，“实现和使用 NBS 数据加密标准指南”在 1981 年发行。NBS 还发表了 FIPS PUB 112，指定 DES 用于口令加密和 FIPS PUB 113，指定 DES 用于计算机数据认证。

DES 的采用

美国国家标准局(ANSI)赞同把 DES 作为私人使用的标准(ANSI X3. 92)，他们称之为数据加密算法(DEA)，ANSI 公布了相应 NBS 文档操作的 DEA 方式的标准(ANSI X3. 106)和一种使用 DES 网络加密的标准(ANSI X3. 105)。

ANSI 内代表零售和批发银行业的二个组，根据 DES 开发了用户标准。零售银行业处理金融机构与私人个体户的业务，批发银行处理金融机构与团体客户的业务。

ANSI 金融机构零售安全性工作组(Financial Instituation Retail Security Working Group)为 PIN 的管理和安全性开发了一种标准(ANSI X9. 8)，为认证零售金融报文开发了另一种基于 DES 的标准(ANSI X9. 19)，该组有一个安全密钥分发的标准草案(ANSI X9. 24)。

ANSI 金融机构批发安全性工作组(Financial Instituation Wholesale Security Working Group)为报文认证(ANSI X9. 9)、密钥管理(ANSI X9. 17)、加密(ANSI X9. 23)以及安全的身份和节点认证(ANSI X9. 26)开发了另一组标准。

美国银行家协会为金融业推出了一个密钥管理的标准，并推荐在任何需要加密时使用 DES 的标准。

在 1987 年制定计算机安全条例(Computer Security Art)之前，普通服务管理局(General Sevices Administration, GSA)负责开发联邦电信标准，自从颁布计算机安全条例以来，这项任务转交给 NIST。GSA 公布了三个使用 DES 的标准：两个是针对一般安全性和相互操作性需求(联邦标准 1026 和联邦标准 1027)，一个是针对 Group 3 传真设备(联邦标准 1028)。

美国财政部制定了一项政策指令，要求所有的电子资金转移报文都要用 DES 认证，他们也制定了基于 DES 要求所有认证设备都必须满足的准则。

国际标准化组织(ISO)首先表决赞同 DES(更名为 DES-1)作为国际标准，并决定不干涉密码术的标准化过程，进而，ISO 的国际批发金融标准(International Wholesale Financial Standards)组在国际认证标准和密钥管理中采用 DES。

DES 装置的生效和证明

作为该标准的一部分，DES NIST 批准了用硬件和固件实现 DES，并已证实其结果遵循了该标准。纯软件实现结果未能生效。因为按照标准，该算法不能以纯软件实现。即使如此，在

通用微处理器上至少有了二种 DES 实现生效：有时在硬件和软件之间划清界线是比较困难的。到 1991 年 5 月 7 日为止，已经有 45 种不同的实现成果，每年约有 3 项新的实现结果生效。

联邦标准 1027 包含了超过 FIPS 46，FIPS 74 和 FIPS 81 的另外要求，为满足该标准，DES 硬件必须安置在物理访问控制之内，包括锁和警报，并且该装置必须设计成故障时不会泄露敏感数据。NSA 证明该装置满足联邦标准 1027 的要求。到 1991 年为止，至少有 32 种产品满足该标准。

对于认证，NIST 设计了一个程序证明该装置符合 ANSI X9.9 和 FIPS 113。到 1991 年 5 月 7 日为止，已经有 29 种产品生效。财政部有另外一个证明程序。NIST 目前正在设计一个程序用于使该装置符合 ANSI X9.17 的密钥管理。

1987 年

该标准规定每五年审查一次，NBS 在 1987 年 3 月 6 日的“联邦注册”上发布了第二个“五年审查”的征集评论告示，NBS 提供三种考虑选择：重申该标准再使用五年，撤消该标准或修改该标准的适用性。

NBS 和 NSA 审查该标准，这一次 NSA 更为专注，因为有里根总统签字的执行指令，称为 NSDD-145，NSA 有权否决 NBS 在密码界的作用。最初，NSA 声明不再为该标准做出保证。DES 的问题不在于已被破译，或者怀疑已被破译，而是人们似乎越来越认为它是能被破译的。

NSA 提出了商业 COMSEC 保证程序(Commercial COMSEC Endorsement Program，CCEP)，该程序最后提供一系列算法来代替 DES，这些 NSA 设计的算法是不会公开的。

这个声明的反响并不太好，许多人指出商业(尤其是金融业)广泛采用 DES，并且没有足够的替代物可选，撤消该标准会使许多机构的数据没有保护。几番争论之后，重申 DES 在 1992 年之前仍为美国政府标准，NBS 不再支持 DES。

1992 年

在 1992 年还是没有 DES 的替代物，NBS，现在称为 NIST，再次在“联邦注册”上征求 DES 的评论：

“本告示的目的是声明评价延续采用该标准保护计算机数据，在下列 FIPS 46－1 的替代物中征求各行业和公众的评论，评论中应包括替代物的费用(影响)和好处：

——重申该标准再使用 5 年。NIST 应继续使实现该标准的装置生效。FIPS 46－1 应该继续是保护非保密计算机数据的唯一赞同的方法。

——撤消该标准。NIST 应不再支持该标准，各机构可以继续使用实现该标准的现有装置。NIST 可以发布其它标准替代 DES。

——修正该标准的适用性和实现方法。这些修正包括改变标准以允许 DES 软件实现，允许 DES 在特定应用场合迭代使用，允许 NIST 赞同和注册使用其它算法。”

评论期于 1992 年 11 月 10 日结束。即使技术评估办公室(Office of Technology Assessment)引述 D. Branstead 认为 DES 的有效生命期将于 90 年代末结束，算法还是再批准了。按照 NIST 代理主管的陈述：

“去年，NIST 正式征求再批准 DES 的评论。在审视了这些评论和我所收到的其它技术输入之后，我打算劝告商业部长再批准 DES 使用 5 年，我也打算建议部长当我们宣布再批准时申明我们的意图：在后 5 年内考虑其替代物。通过这个宣布，我们希望给人民一个机会去评论有次序的科技转变。同时，我们需要考虑依赖于这个已经证明的标准的系统的广大基础。”

直至 1996 年底 DES 还是沿用为加密标准，征求替代物的工作仍在延续。

DES 算法概况

DES 是一种块密码:它以 64 比特的块加密数据,算法输入明文的 64 位的块,输出为 64 位的密文。加密和解密用同一算法。

密钥为 56 位(密钥通常表示成 64 位,每字节的第 8 位用作奇偶校验位),密钥可以是任一个 56 比特的数并可随时更换。有的数被认为是弱钥,但可以避免采用,安全性全在于密钥。

在最简单的层次上,算法仅仅是两个基本加密技术:混乱和扩散的组合。DES 构造块的基本方法是这些技术(置换后跟随一个置换)根据密钥在文本上的一个组合,这称为一轮(round)。DES 有 16 轮:在明文块上使用同一技术组合 16 次(见图 10.1)。

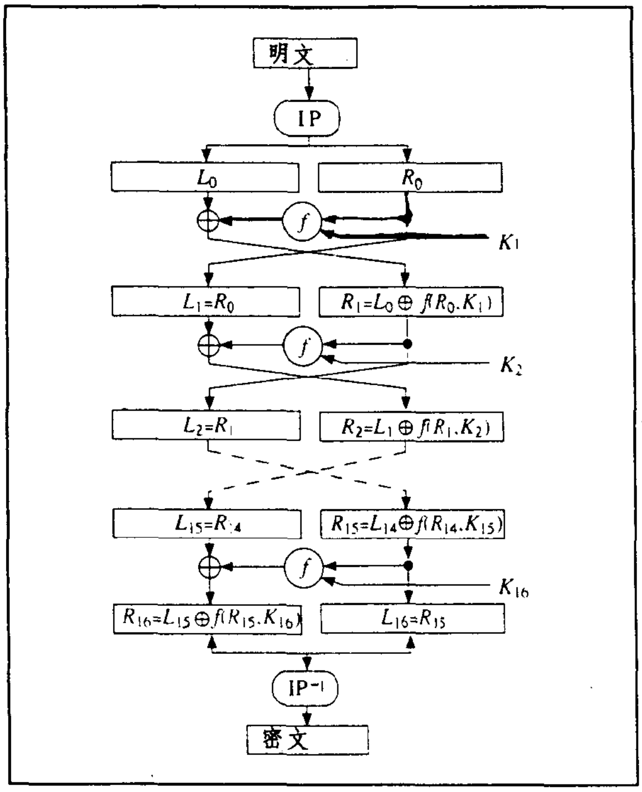


图 10.1 DES 结构

算法仅使用了至多 64 比特数的标准算术和逻辑运算,因此易于在 70 年代末的硬件技术上实现,最初的软件实现是拙劣的,但目前的实现已有许多改进,算法的重复特征适合于在专用芯片上实现。

DES 算法概要

DES 对明文的 64 位块操作,在初始置换 IP(Initial Permutation)之后,64 位的块分为各为 32 位的左半部分和右半部分,接着是 16 轮同样的操作,称为函数 f,其中将数据与密钥结合。在第 16 轮之后,右半部分与左半部分联合,再经过初始置换的逆 IP^{-1}结束算法。

在每一轮中,密钥先移位,再从 56 位密钥中选取 48 位。数据右半部分通过扩展排列(expansion permutation)由 32 位扩展为 48 位,再与经移位和置换的 48 位密钥异或结合,其结果用某一替代算法替代 32 位新数据,并再次置换。这四个操作构成函数 f,函数 f 的输出再与左半部分异或结合,其结果作为新的右半部分,而原来的右半部分成为新的左半部分。这些操作重复 16 次,形成 DES 的 16 轮(见图 10.2)。

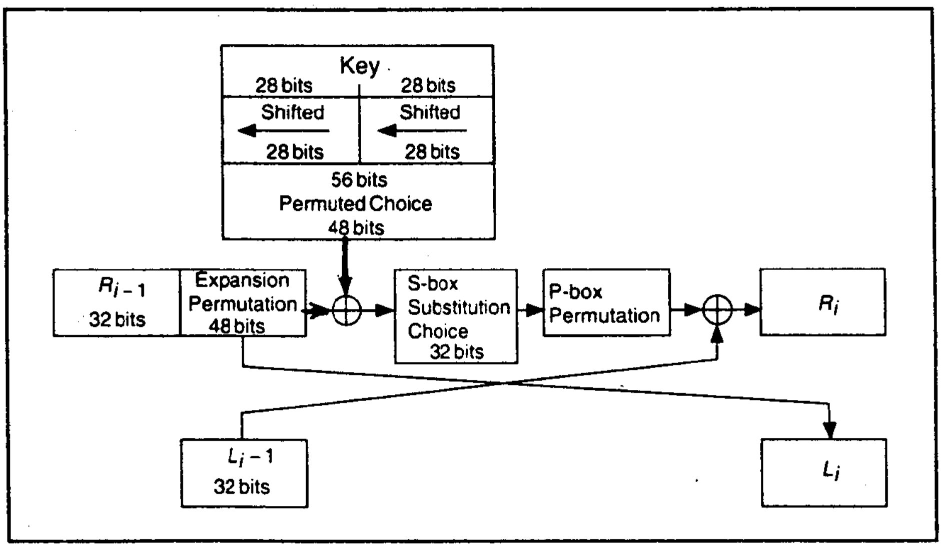


图 10.2 DES 算法一轮的详细过程

如果 B_i 为第 i 次迭代的结果，L_i 和 R_i 各为 B_i 的左半部分和右半部分，K_i 为第 i 轮的 48 位密钥，f 为执行所有替代、排列、与密钥异或操作的函数，则一轮为：

$$L_i = R_{i-1} \qquad\qquad i=1,2,\cdots,16$$

$$R_i = L_{i-1}\ \mathrm{XOR}\ f(R_{i-1}, K_i)$$

初始置换

初始置换是把输入块的 64 位打乱重排，如表 10.1 所示。该表与本章中的其它表一样，都是从左到右、从上到下的顺序排列。置换后的 64 位数据的第 1,2,…,64 位依次是原输入数据的第 58,50,…,7 位。

表 10.1 初始置换

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

初始置换和最后的逆初始置换不影响 DES 的安全性，由于这种位置换方式以软件实现是比较困难的(虽然以硬件实现是轻而易举的)，许多 DES 的软件实现都不使用初始置换及其逆置换。虽然这种算法不比 DES 的安全性差，但它没有遵循 DES 的标准而不能称为 DES。

密钥变换

最初，64 位密钥每隔 8 位删除 1 位校验位即成为 56 位，如表 10.2 所示。选择出 56 位密钥之后，DES 的每轮都产生一个不同的 48 位密钥 K_i，产生方法如下所述。

表 10.2 密钥变换

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36
63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

首先，56 位的密钥分成各为 28 位的二半部分，每一部分都根据各轮循环左移 1 位或 2

位。移位位数见表 10.3。

表 10.3　各轮的移位位数

轮号	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
移位数	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

移位之后，从 56 位密钥中选择 48 位。由于这个操作置换位的顺序同时又选择一个位的子集，称为压缩排列(compression permutation)或选排列(permuted choice)，选排列结果与扩展排列结果的大小相同。表 10.4 给出选排列一例，移位后密钥中的第 33 位移到了输出中的第 35 位，而第 18 位删除掉了。

表 10.4　选排列

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

扩展排列

扩展排列把数据的右半部分 R_i 由 32 位扩展到 48 位，扩展排列置换了位的次序同时也重复了某些位。扩展排列的目的有二个：为异或运算使得其结果与密钥大小相同，产生一个长的结果而后在替代操作时又可将其压缩。

不过扩展排列的目的并不是主要的密码目的，通过允许 1 位影响 2 次替代操作，输出位从属于输入位扩展得更快，这称为雪崩准则(avalanche criteria)。许多 DES 的设计都尽快地使密文的每一位依赖于明文的每一位和密钥的每一位。

图 10.3 定义了扩展排列的过程，有时也称之为 E 盒。对每一个 4 位输入组，第 1 位和第 4

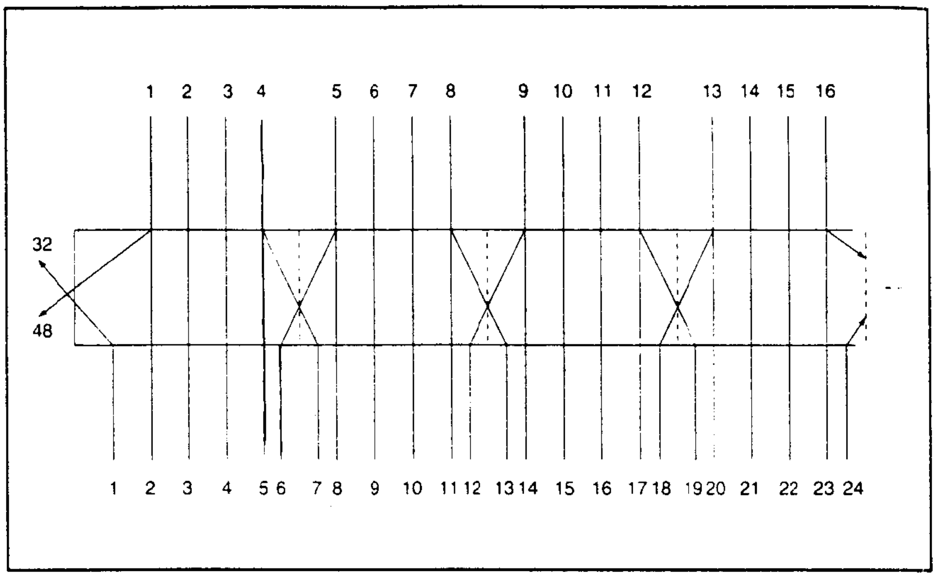


图 10.3　扩展排列过程

位表示输出组中的 2 位，而第 2 位和第 3 位表示输出组中的 1 位，表 10.5 给出输出位与输入位之间的对应关系，例如输入组中的第 3 位移到输出组中的第 4 位，输入组中的第 21 位成为

输出的第 30 位和第 32 位。

表 10.5　扩展排列

32	1	2	3	4	5	4	5	6	7	8	9
8	9	10	11	12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21	22	23	24	25
24	25	26	27	28	29	28	29	30	31	32	1

虽然输出块比输入块大，但只有一个输入块能生成特定的输出块。

S 盒替代

压缩的密钥与扩展的块异或之后，48 位的结果开始执行一个替代操作，替代由 8 位替代盒(substitution box)或称为 S 盒完成。48 位数据分成 8 个 6 位组，每一组由一个 S 盒完成操作。

每一个 S 盒是个 4 行 16 列的表，表中的每一项为一个 4 位数，S 盒的 6 位输入指定在哪一行和哪一列查找输出结果，表 10.6 给出 8 个 S 盒。

表 10.6　S 盒替代

第一个 S 盒															
14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
第二个 S 盒															
15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
第三个 S 盒															
10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
第四个 S 盒															
7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
第五个 S 盒															
2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3
第六个 S 盒															
12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
第七个 S 盒															
4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	5	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

续上表

第八个S盒															
13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

S 盒的输入以非常特殊的方式确定 S 盒中的项。记 S 盒的 6 位输入为 b_1, b_2, b_3, b_4, b_5 和 b_6，位 b_1 和 b_6 组成 2 位的二进制数 b_1b_6，其取值为 0～3，对应于表中的行号。中间的 4 位构成一个 4 位的二进制数 $b_2b_3b_4b_5$，其取值为 0～15，对应于表中的列号。

例如，假设第 6 个 S 盒的输入(异或结果的 31～36 位)为 110010，第 1 位和最后 1 位构成 10，对应于第 6 个 S 盒的第 2 行。中间的 4 位构成 1001，对应于同一个 S 盒的第 9 列。S 盒中第 2 行第 9 列的项是 14，所以用 1110 替代 110010。

S 盒是算法的关键性步骤，算法中其它所有操作都是线性的，易于分析，而 S 盒是非线性的，为 DES 提供了安全性。

替代阶段的操作结果为 8 个 4 位组，重新组合成一个 32 位的块，该块开始执行下一个操作：P 盒置换。

P 盒置换

S 盒替代的 32 位输出由 P 盒作置换操作，这个置换把每个输入位移到输出中的某一位置：其中没有一位使用二次也不删除任何位，这称为直接置换(straight permutation)或就称为置换。表 10.7 给出每一位移到的位置，例如，第 4 位移到第 21 位，第 31 位移到第 4 位。

表 10.7 P 盒置换

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

最后，P 盒置换的结果与原先 64 位块的左半部分异或，再左半部分和右半部分交换并开始下一轮处理。

最后置换

最后置换是初始置换的逆，如表 10.8 所示。注意在 DES 的最后一轮处理之后左半部分与右半部分没有交换，而把连接块 $R_{16}L_{16}$ 作为最后置换的输入，把二部分交换并循环移位将产生完全一样的结果，这就是为什么该算法即可用于加密又可用于解密。

表 10.8 最后替换

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

DES 的解密

在所有替代、置换、异或和循环移位操作之后，也许会以为解密算法与加密算法完全不同，相反，各种操作都是非常精心地设计的，由此产生一个非常有用的性质，即同一个算法既可用于加密又可用于解密。

在 DES 中使用同一函数作加密或解密是可能的，唯一的区别是解密过程中密钥必须逆序使

用，即如果每一轮的加密密钥为$k_1,k_2,k_3,\cdots,k_{16}$，则解密密钥相应为$k_{16},k_{15},k_{14},\cdots,k_1$。DES 产生用于每一轮的密钥是循环的，密钥移位为右移，并且移位的位数按从右到左的顺序从表中选取。

$$R_{i-1}=L_i \qquad i=16,15,\cdots,1$$

$$L_{i-1}=R_i \text{ XOR } f(L_i,K_i)$$

DES 的方式

FIPS PUB 81 指定了 4 种操作方式：电码本方式、密码块链接方式、输出反馈方式和密码反馈方式。ANSI 银行业标准指定 ECB 和 CBC 方式用于加密，CBC 和 n 位 CFB 方式用于认证。

在软件界，DES 的使用证书并无争议。虽 ECB 方式易受攻击，但由于简单常用于畅销的商品软件中，CBC 比 ECB 稍微复杂，但提供了更高的安全性，偶有采用。

DES 的硬件和软件实现

有关 DES 的硬件和软件有效实现的论文已有很多，包括软件实现中加速算法速度的实用建议。

最快的 DES 芯片的记录保持者是 DES 开发的一个原型，它支持 ECB 和 CBC 方式并基于一个 50K 个晶体管的 GaAs 门阵列，数据可以以 1GBit/S 的速度加密/解密，即每秒转换 $1.56\text{x}10^7$ 块数据。

DES 在 IBM 3090 大型机上的一种软件实现可每秒执行 32,000 次 DES 加密，微机会慢一些。表 10.9 给出对各种 Intel 和 Motorola 微处理器的实际结果和估计值。

表 10.9 不同微处理器上的 DES 速度

处理器	速度(MHz)	总线宽度(bit)	DES 块数(每秒)
8088	4.7	8	370
68000	7.6	16	900
80286	6.0	16	1,100
68020	16.0	32	3,500
68030	16.0	32	3,900
80286	25.0	16	5,000
68030	50.0	32	9,600
68040	25.0	32	16,000
68040	40.0	32	23,200
80486	33.0	32	40,600+

这个测试软件用 C 与汇编编写，代码大小约为 64K ANSI C 的实现将慢 20%

DES 的安全性

人们对 DES 的安全性已经争论了很久，对钥长、迭代次数和 S 盒的设计存在许多疑虑。S 盒尤其神秘，没有任何明显的原因为什么要如此选取所有这些常量。有人担心 NSA 在算法中嵌入某种“陷门”，使得他们可以用方便的手段解密信息。

1978 年美国国会就此事作了调查，调查结果是保密的，但一个非机密的摘要宣布 NSA 并没有不适当地干涉了 DES 的设计，然而，由于政府没有公开调查详情，许多人还是持怀疑态度。IBM 中设计 DES 的二位密码编制人员，Tuchman 和 Meyer，也宣布 NSA 没有修改他们的设计。

弱钥

由于算法的每一轮中的子钥都是由初始密钥分成二半，每一半独立循环移位而得，有的初始密钥是弱钥。如果密钥的每一半部分中的位是全 0 或全 1，则算法中每一轮的密钥都是相同的，这种情况当密钥全 0、全 1、或一半密钥全 0 另一半全 1 时都会出现，同样，其中二个弱钥具有削弱安全性的性质。

表 10.10 给出了 4 个十六进制表示的弱钥(初始置换把每第 8 位抽出作为校验位,并且密钥的次序稍有置乱)。

表 10.10 DES 的弱钥

弱钥值	实际密钥
0101010101010101	0000000000000000
FEFEFEFEFEFEFEFE	FFFFFFFFFFFFFFFF
1F1F1F1F1F1F1F1F	00000000 FFFFFFFF
E0E0E0E0E0E0E0E0	FFFFFFFF 00000000

另外,有一些不同的密钥对能产生相同的密文,即密钥对中的一个密钥能解开用另一个密钥加密的报文。这也是由于 DES 中产生子钥的方式:只产生 2 个不同的子钥而不是产生 16 个不同的子钥,每个子钥在算法中使用 8 次,这些密钥称为半弱钥,表 10.11 给出十六进制表示的半弱钥。

表 10.11 DES 的半弱钥

01FE01FE01FE01FE	FE01FE01FE01FE01
1FE01FE01FE01FE0	E01FE01FF10EF10E
01E001E001E001E0	E001E001F101F101
1FFE1FFE1FFE1FFE	FE1FFE1FFE0EFE0E
011F011F010E010E	1F011F010E010E01
E0FEE0FEF1FEF1FE	FEE0FEE0FEF1FEF1

还有一些密钥只产生 4 个子钥,每个子钥在算法中使用 4 次。表 10.12 列出这些密钥。

表 10.12 DES 可能的弱钥

1F1F01010E0E0101	E00101E0F10101F1
011F1F01010E0E01	FE1F01E0FE0E01F1
1F01011F0E01010E	FE011FE0FE010EF1
01011F1F01010E0E	E01F1FE0F10E0EF1
FE0101FEFE0101FE	
E0E00101F1F10101	E01F01FEF10E01FE
FEFE0101FEFE0101	E01F01FEF10E01FE
FEE01F01FEF10E01	FE1F1FFEFE0E0EFE
E0FE1F01F1FE0E01	
FEE0011FFEF1010E	1FFE01E00EFE01F1
E0FE011FF1FE010E	01FE1FE001FE0EF1
E0E01F1FF1F10E0E	1FE001FE0EF101FE
FEFE1F1FFEFE0E0E	01E01FFE01F10EFE
FE1FE001FE0EF101	0101E0E00101F1F1
E01FFE01F10EFE01	1F1FE0E00E0EF1F1
FE01E01FFE01F10E	1F01FEE00101FEF1
E001FE1FF101FE0E	011FFEE0010EFEF1
1F01E0FE0E01F1FE	
01E0E00101F1F101	011FE0FE010EF1FE
1FFEE0010EFEF001	0101FEFE0101FEFE
1FE0FE010EF1FE01	1F1FFEFE0E0EFEFE
01FEFE0101FEFE01	
1FE0E01F0EF1F10E	FEFEE0E0FEFEF1F1
01FEE01F01FEF10E	E0FEFEE0F1FEFEF1
01E0FE1F01F1FE0E	FEE0F0FEFEF1F1FE
1FFEFE1F0EFEFE0E	E0E0FEFEF1F1FEFE

诚然，弱钥的存在是 DES 的不足，但将上述列出的 64 个弱钥与密钥总数的 72,057,594,037,927,936 相比仍是微不足道的，如果随机选择密钥，选中弱钥的可能性是可以忽略的。

对弱钥和半弱钥曾作了进一步的分析，但没有发现有其他弱点的密钥模式。

互补对称性

取密钥的补是指将所有的 0 替换成 1，将 1 替换成 0。如果原密钥加密某明文，则密钥的补可以把明文的补加密成密文的补。

记/x 为 x 的补，则

$E_k(P) = C$

$E_{/k}(/P) = /C$

互补对称性会使 DES 在选择明文攻击时所需的工作量减半，只需测试 2^{55} 个可能密钥而不是 2^{56}。Biham 和 Shamir 证明了同样复杂度的已知明文攻击至少需要有 2^{33} 个已知明文。

互补对称性是不是 DES 的弱点颇值得讨论，由于大多数报文都没有互补的明文块（在随机明文中，可能性还是很高的），告诫用户不要使用互补密钥。

DES 是不是群？

把所有可能的 64 位明文块转换为所有可能的 64 位密文块的不同方法有 2^{64} 种，DES 算法以 56 位密钥有 $2^{56}(10^{16})$ 种转换方法，采用多重加密似乎可能达到更多种转换方法，然而，这只有当 DES 操作不是群时才成立。

例如，整数相加是一个群，二个整数相加之和为另一整数，而整数相乘除不是群，二个整数相乘为另一整数，但二个整数相除有时为分数。

如果 DES 是一个群，则密码分析就容易得多，多重加密就没有作用了，因为总归存在一个密钥 k_3，使得 $E_{k_2}(E_{k_1}(P)) = E_{k_3}(P)$，即先用 k_1 再以 k_2 加密明文块等效于以 k_3 加密，更坏的是，DES 对中间会合攻击是敏感的，只需执行 2^{28} 步，远远小于穷举搜索所需的 2^{56} 步。

许多数学家都对这个问题作了研究，这些经验性的结果汇集成"势不可挡的证据"说明 DES 不是群，但直到 1992 年还没有证明 DES 不是群。

钥长

最初 IBM 设计 Lucifer 时的钥长为 128 位，当 DES 成为标准时有效长度缩减为 56 位，要求增加钥长的争论主要是针对蛮力攻击的可能性。

1979 年，Diffie 和 Hellman 争论说一种专用攻击 DES 的并行计算机耗资 2000 万美元可在 1 天内找到密钥，1981 年，Diffie 把它修正为费用 5000 万美元、时间为 2 天。Diffie 和 Hellman 争议说虽然除 NASA 之类的机构外是支付不起这笔费用的，但到 1990 年 DES 将彻底不安全。

Hellman 对短密钥提出另一种意见：通过以存储空间换取时间可能加快搜索过程，他提出可以在每一可能密钥下计算并存储加密某一明文块的 2^{56} 种可能结果，接着攻破一未知的密钥只需密码分析者把明文块插入到加密流中，得到结果密文，然后通过核对结果（查表）确定密钥。Hellman 估计这种破译机当前费用为 500 万美元。

同时，DES 的硬件实现慢慢接近 Diffie 和 Hellman 专用机的每秒 100 万次加密的需求，1984 年已生产出每秒能执行 256,000 次加密的 DES 芯片，1987 年开发出了每秒执行 512,000 次加密的芯片，并且每秒核对 100 多万个密钥的芯片也问世了。随着硬件性能的改进及价格的下降，Hellman 指出实现这样的机器将是可能的。

直到 1990 年二位以色列数学家，Biham 和 Shamir，提出了差分密码分析（differential

cryptanalysis)技术才平息了有关钥长的争论。在讨论这一技术之前,先介绍一些对 DES 的其它批评。

迭代次数

16 轮迭代是否足够? A. Konheim 说明了 8 轮迭代之后,密文基本上就是每一明文位和密钥位的随机函数,为什么算法不在 8 轮之后停止?

在过去几年中,已经成功地攻破了少量轮数的 DES 变例,1982 年就可轻易地攻破了 3 轮或 4 轮的 DES,几年之后又攻破了 6 轮的 DES。Biham 和 Shamir 的差分密码分析解释了对于小于 16 轮的 DES 用已知明文攻击比蛮力攻击更为有效。

S 盒的设计

除被谴责缩减了钥长之外,NSA 也被谴责修改了 S 盒的内容。当要求说明 S 盒设计的正确性时,NSA 表示算法设计的关键部分是"敏感的",不得公开。许多密码编制人员担心 NSA 设计的 S 盒中隐藏了陷门,使得只有他们可以分析这个算法。

从此,许多人投入到分析 S 盒的设计及操作。在 70 年代中期,Lexar 公司和 Bell 实验室研究了 S 盒的操作过程,虽然 Bell 实验室的人员声称 S 盒内可能隐藏有陷门,但是谁也没有发现任何弱点。

在第二次 DES 研讨会上,NSA 公布了 S 盒的一些设计准则:

1. 没有一个 S 盒是其输入的线性或仿射函数,即其 4 个输出位不能表示成其 6 位输入的线性方程式

2. 改变 S 盒的 1 位输入将至少引起 2 位输出变化,即 S 盒能很好地将信息扩散到所有的输出位

3. 当任一输入位保持不变时,S 盒被选择为使 0 和 1 的个数之差最小,即如果固定输入的某一位,然后改变其它 5 位,不应该导致输出的 0 和 1 的数目相差很大。

这没有根本消除人们的疑虑,争论继续进行。

从此,又有许多人从事研究 S 盒的设计。第四个 S 盒 S4 有一些有趣的性质:最后三个输出位和第一位一样可以通过对某些输入位求补而导出。S 盒的二个不同的、但精心选择的输入可以产生同样的输出,通过改变 3 个邻近 S 盒的某些位可以得到某个 S 盒的相同输出。

差分密码分析

1990 年,E. Biham 和 A. Shamir 提出了差分密码分析的概念,用这种方法,Biham 和 Shamir 发现 DES 的一种选择明文攻击比蛮力攻击有效得多。

差分密码分析检查明文对和密文对,这种攻击特别检查明文之间具有特殊差分的密文对。二个明文可以随机选择,长度满足特殊的差分条件,并且密码分析者也无需知道其值。

有的明文对中的差分很可能在生成密文对中重复出现,这些差分称为特征,例如,如果二个明文对之间的差分以十六进制表示为 0080 8200 6000 0000,则(忽略 DES 的初始置换)在 3 轮之后有$(1/16)^2$ 或 5%的可能性在结果密文对之间的差分还是 0080 8200 6000 0000(见图 10.4)。Biham 和 Shamir 发现了许多类似的特征。

差分密码分析使用的这些特征为可能的密钥赋以概率,最后定出最高概率的密钥,这是个统计攻击方法并且成功率相当高。

这种方法可以攻击 DES 和其它具有常量 S 盒的类似算法,这种攻击严重地依赖于 S 盒的结构:DES 中的 S 盒恰好对差分密码分析作了优化。

表 10.13 给出攻击不同轮数的 DES 的概要。

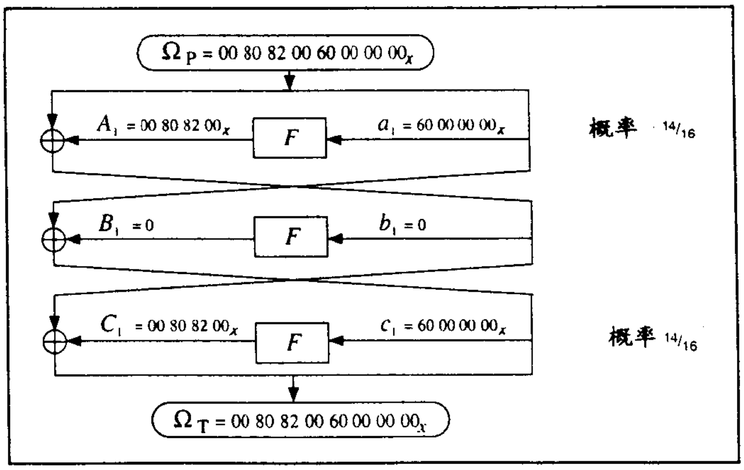


图 10.4 具有 5%可能性的 DES 的 3 轮特征

表 10.13 对 DES 的差分密码分析攻击

轮数	选择明文	已知明文	分析明文	分析复杂度
8	2^{14}	2^{38}	4	2^{9}
9	2^{24}	2^{44}	2	$2^{32}+$
10	2^{24}	2^{43}	2^{14}	2^{15}
11	2^{31}	2^{47}	2	$2^{32}+$
12	2^{31}	2^{47}	2^{21}	2^{21}
13	2^{39}	2^{52}	2	$2^{32}+$
14	2^{39}	2^{51}	2^{29}	2^{29}
15	2^{47}	2^{56}	2^{7}	2^{37}
16	2^{47}	2^{55}	2^{36}	2^{37}

+通过使用近 4 倍的明文可以极大地降低分析的复杂度

可以通过增加迭代次数来改进 DES，对 17 或 18 轮的 DES 作差分密码分析（选择明文）基本与穷举搜索的复杂度相当，对 19 轮的 DES，穷举搜索比差分密码分析容易。

有一些问题值得注意。首先这种攻击主要是理论上的，差分密码分析所需的大量时间和数据几乎无人可及，为得到攻击所需的数据，几乎需要 3 年加密 1.5M/S 的选择明文数据流。其次，这是个选择明文攻击，差分密码分析也以已知明文攻击的方法操作，但需要在所有的明文密文对中精选有用的数据。对于 16 轮的 DES，这将使差分密码分析比蛮力攻击稍稍复杂（差分密码分析需要 $2^{(55-1)}$ 次操作，蛮力攻击为 2^{55}），因此现在普遍认为恰当地实现的 DES 对差分密码分析还是安全的。

相关密钥密码分析

表 10.3 给出了 DES 的密钥在每轮中的循环左移位数，除第 1，2，9 和 16 轮中移动 1 位之外，其它各轮都移动 2 位，这是为什么？

有一种修改的 DES，其中每轮中密钥都循环左移 2 位，它的安全性要比 DES 差。E. Biham 设计了一种相关密钥密码分析攻击方法，可以用 2^{17} 个选择密钥的选择明文，或者 2^{23} 个选择密钥的已知明文攻破这种 DES 的变例。

相关密钥密码分析（related-key cryptanalysis）类似于差分密码分析，检查不同密钥之间的差分，分析者选择一对密钥之间的关系也不是选择密钥本身。数据以二个密钥加密。在相关密钥密码分析的已知明文方式中，分析者知道明文和以密钥加密随机数据而得的密文，在选择明文方式中，分析者选择以二个密钥加密的明文。

这种攻击方法与密码算法的轮数无关，对 16 轮、32 轮、甚至 1000 轮的 DES 攻击的复杂性

是相同的。

线性密码分析

M. Matsui 提出另一种对 DES 的已知明文攻击方法，称为线性密码分析(linear cryptanalysis)，它用 2^{21}个已知明文可以攻破 8 轮的 DES，用 2^{43}个明文攻破 16 轮的 DES。这是迄今为止对 DES 最有效的攻击。

10.2 DES 的变例

多重 DES

有的 DES 实现中采用三重 DES(如图 10.5)，由于 DES 不是群，则用穷举搜索攻击结果密文是很难的，需测试 2^{112}次而不是 2^{56}次。

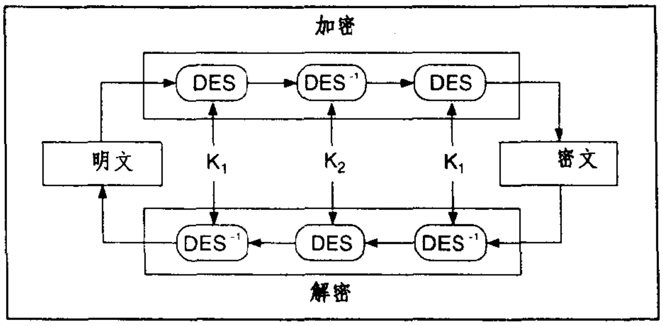


图 10.5 三重 DES

独立子钥的 DES

这种变例中每轮使用不同的子钥，而不是从单一的 56 位密钥中产生。由于 16 轮中每轮都需要 48 位密钥，所以这种变例的钥长相当于 768 位，这将大大提高了 DES 的复杂性。

Biham 和 Shamir 证明了这种变例可以用 2^{61}个选择明文攻击，而不是蛮力攻击所需的 2^{768}位(见表 10-14)，这说明密钥安排的任何修改不会使 DES 强多少。

表 10.14 对 DES 各种变例的差分密码分析攻击

修改的操作	选择明文	修改的操作	选择明文
完全的 DES(无修改)	2^{47}(不独立密钥)	一项	2^{33}
P 排列	没有增强	统一表	2^{26}
同一性置换	2^{19}	取消 E 排列	2^{26}
S 盒的次序	2^{38}	E 和子钥异或的次序	2^{44}
加法替代异或	2^{39}, 2^{3}	GDES(q=8)	
S 盒		16 轮	6,16
随机	2^{21}	64 轮	2^{49}(独立密钥)
随机置换	$2^{44}-2^{48}$		

交替 S 盒的 DES

其它 DES 的修改集中在 S 盒上，有的设计使 S 盒的次序根据密钥而可变，也有的设计中 S 盒的内容可变。Biham 和 Shamir 证明了 S 盒的设计及其次序都已对防备差分密码分析作了优化(见表 10.14)。

“8 个 DES 的 S 盒的次序更换(内容不变)使 DES 更脆弱：以某种更换次序的 16 轮 DES 在 2^{38}步内可破…随机 S 盒的 DES 非常易于攻破。即使 DES 的 S 盒中的一项微小改变也使 DES 更易于攻破。”

但还没人能够证明关于所有已知攻击没有更好的 S 盒。

可变 S 盒的 DES

只有分析者知道 S 盒的构造时才能作差分分析，如果 S 盒是根据密钥而可变的且保密，那就无法作差分分析。

CRYPT(3)

CRYPT(3)是 UNIX 系统 V 上 DES 的变例，它主要用作口令的单向函数，有时也用于加密。这种变例是具有密钥独立的 S 盒和 E 盒的 DES。

广义的 DES

广义的 DES(Generalizied DES, GDES)是为加速 DES 和增强算法的强度而设计的，整个块的大小增加了，而函数 f 保持不变。

图 10.6 为 GDES 的块算法。GDES 可处理可变大小的明文块，加密块分成 q 个 32 位的子块，具体的子块数取决于整个块的大小(设计中这是可变的，但实现时必须固定)，一般 q 等于块大小除以 32。

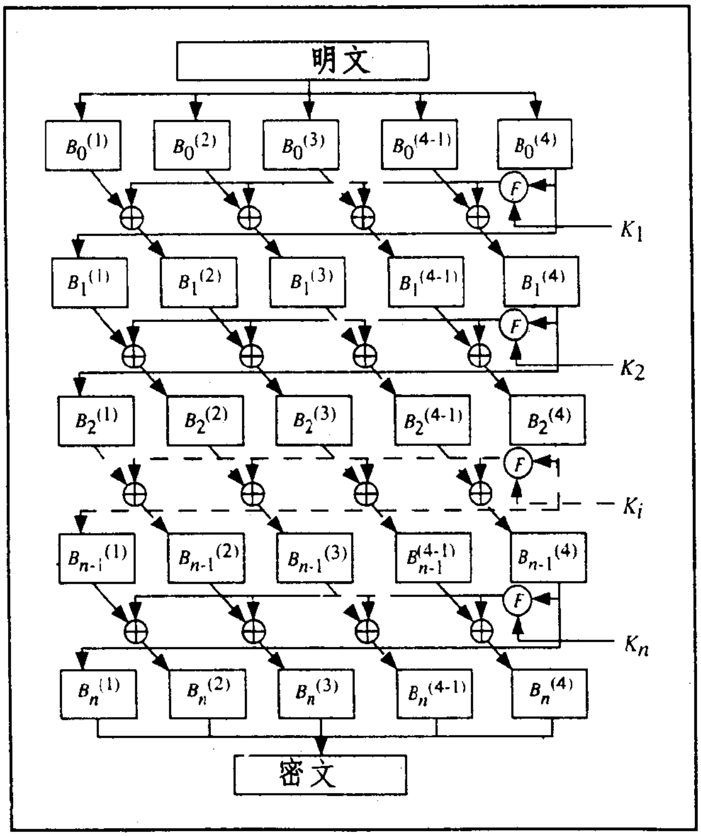


图 10.6 GDES 概况

函数 f 在最右边的块中每轮计算一次，结果与所有其它部分异或，并将异或结果循环右移。GDES 的轮数 n 可变，最后一轮作了一点修改，使得加密和解密操作只有子钥的次序不同(类似于 DES)。事实上，如果 q=2 且 n=16，则 GDES 就是 DES。

Biham 和 Shamir 证明了对于 q=8 且 n=16 的 GDES，只需 6 个选择明文就可用差分密码分析攻破。如果采用了独立子钥，也只需 16 个选择明文。对于 q=8 且 n=22 的 GDES，需要 48 个选择明文；对于 q=8 且 n=31 的 GDES，需要 500,000 个选择明文。即使 q=8 且 n=64 的 GDES 也比 DES 脆弱：攻破它需要 2^{49} 个选择明文。事实上，任何 GDES 的方法都比 DES 快但安全性差(见表 10-14)。

第 11 章　其它块算法

11.1　LUCIFER 算法

Lucifer 是 IBM 在 70 年代初设计的块算法，它也是 IBM 研制密码算法的项目的名称。

开始时，项目定义了一个替代-置换网络，早期的论文没有定义太多的算法参数置换、S 盒、密钥安排等，但已给出大概，有较大的参考价值。DES 中函数 f 的输出与前一轮的输入异或，异或结果作为下一轮的输入。Lucifer 中，S 盒的输入是前一轮 S 盒的位置换输出，其中第一轮 S 盒的输入为明文。密钥中的一位用于在二个 S 盒中选取一个。不同于 DES，各轮之间没有交换数据，也没有将块取半而用。

后来他们设计出一种更好的变例，与以前的有很大的区别而更类似于 DES。它完全确定了 16 轮、128 位的块、以及完全定义了 S 盒和密钥安排方法。Lucifer 的一个中间版经修改成为 DES 算法。

用差分密码分析攻击 Lucifer 的第一版，Biham 和 Shamir 证明了具有 32 位块的 8 轮的 Lucifer 可以用 40 个选择明文在 2^{29}步内攻破，对 128 位块的 8 轮的 Lucifer 可以用 60 个选择明文在 2^{53}步内攻破。另一种差分分析法可用 24 个选择明文在 2^{21}步内攻破 128 位块的 18 轮的 Lucifer。所有这些攻击采用强的 DES S 盒。用差分分析法攻击第二版 Lucifer，发现 S 盒要比 DES 的脆弱得多。

相关密钥密码分析可以用 2^{23}个选择密钥的选择明文或 2^{56}个选择密钥的已知明文攻破任何轮数的 128 位 Lucifer。

Lucifer 具有多项美国专利，现在都已期满。

11.2　MADRYGA 算法

W. E. Madryga 在 1984 年提出这个块算法，因为其中没有置换操作，并且所有的操作都是对 8 位的数据块处理的，它可以用软件高效实现。

Madryga 的设计目标是很值得重述的：

1. 没有密钥时不能从密文导出明文(这是指算法是安全的)。

2. 从一些明文和密文中确定密钥所需的操作次数应等于(统计意义上)加密的操作次数与可能的密钥之乘积(这是指应该没有比蛮力攻击更好的明文攻击方法)。

3. 对算法的了解不能挫败密码的强度(这是指安全性应完全在于密钥)。

4. 对于同一个明文，改变 1 位密钥应引起密文的巨大改变；对于同一密钥，改变 1 位明文应引起密文的巨大改变。这称为雪崩效应。

5. 算法应包含替代和置换的不可交换的组合。

6. 算法应包含受输入数据和密钥控制的替代和置换操作。

7. 密文应完全掩盖明文中的冗余位。

8. 密文应与明文等长。

9. 任何密钥与密文结果之间应没有简单的联系。

10. 任何密钥都应产生强的密文(这是指应没有弱钥)。

11. 密钥和正文的长度应可调节,以满足各种应用场合和安全性的需求。

12. 算法应可以在大型机、小型机、微机及离散逻辑上高效实现(事实上,算法中使用的功能只限于异或和移位)。

DES 已经实现了 1 到 9 个目标,但后 3 个没有实现。假设攻破算法的最好方法是蛮力攻击,可变长的密钥当然能平息认为 56 位太少的争论,他们可以用任何钥长实现算法。对于曾试图以软件实现 DES 的人来说,考虑软件实现的算法是受欢迎的。

Madryga 算法描述

Madryga 算法由二层循环嵌套构成,外层循环重复 8 次(如果要保证安全性则可增加重复次数),并含有一个对明文处理的内层循环。内层循环实现将明文转换为密文,对明文以一个 8 比特块(1 个字节)重复。这样,算法对整个明文连续扫描 8 次。

内层循环中对 3 字节的数据窗口的一次迭代称为工作帧(working frame),如图 11.1 所示。每次迭代窗口前进 1 个字节(处理最后 2 个字节时把数据当成循环的),工作帧的前 2 字节一起循环移位,移位位数可变,而后 1 字节与密钥的某些位异或。当工作帧前进时,所有字节依次循环移位以及与密钥异或,依次的循环移位重叠了前面异或和移位的结果,并且异或结果作用于移位,这使整个过程可逆。

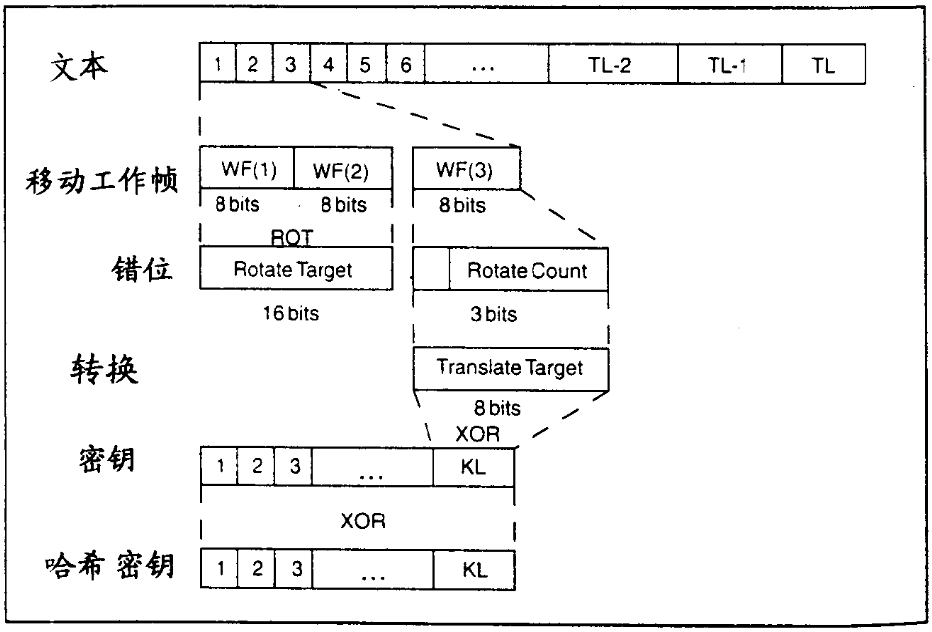


图 11.1 Madryga 的一次迭代

由于每个字节的数据影响其左边 2 字节和右边 1 字节数据,所以外层循环 8 次以后,密文的每个字节依赖于其左边 16 字节和右边 8 字节。

加密时内层循环的每次迭代都从工作帧在明文的倒数第 2 字节开始,并且工作帧循环前进直到明文的倒数第 3 字节为止。首先,整个密钥与一随机常量异或且循环左移 3 位,保存工作帧的低字节的低 3 位以用于后面控制另 2 个字节的循环移位。接着,工作帧的低字节与密钥的低字节异或,2 个字节循环左移 0～7 位。最后,工作帧右移 1 位并且整个过程重复。

随机常量是为了把无论什么密钥转化为伪随机序列,其长度必须与钥长相同。相互通讯

双方所使用的随机常量必须等长。对于 64 位的密钥，Madryga 推荐的常量为 F1E2D3C4B5A6978(十六进制)。

解密是其逆过程，内层循环的每次迭代从工作帧在密文的倒数第 3 个字节开始，并且工作帧反向循环前进直到密文的倒数第 2 个字节为止，密钥和 2 个密文字节右移，并且在循环移位之前执行异或操作。

Madryga 密码分析

Queenland 理工大学的研究人员审查了 Madryga 算法，他们发现 Madryga 算法没有明文-密文雪崩效应，另外，在许多测试密文中 1 的比例比 0 高。

E. Biahm 对 Madryga 算法作了粗略研究，发现其安全性并不太好。Madryga 算法只由线性操作(循环移位和异或)组成，只是根据数据稍稍作了修改，其中没有类似 DES 中 S 盒之类的强度。另外，明文和密文所有位的奇偶性是个常量，只取决于密钥，这样，如果有一个明文及其相应的密文，就可对任何明文预测密文的奇偶性。

11.3 NEWDES 算法

NewDES 是 R. Scott 在 1985 年作为 DES 替代品而设计的，它对 64 位的明文块作处理，

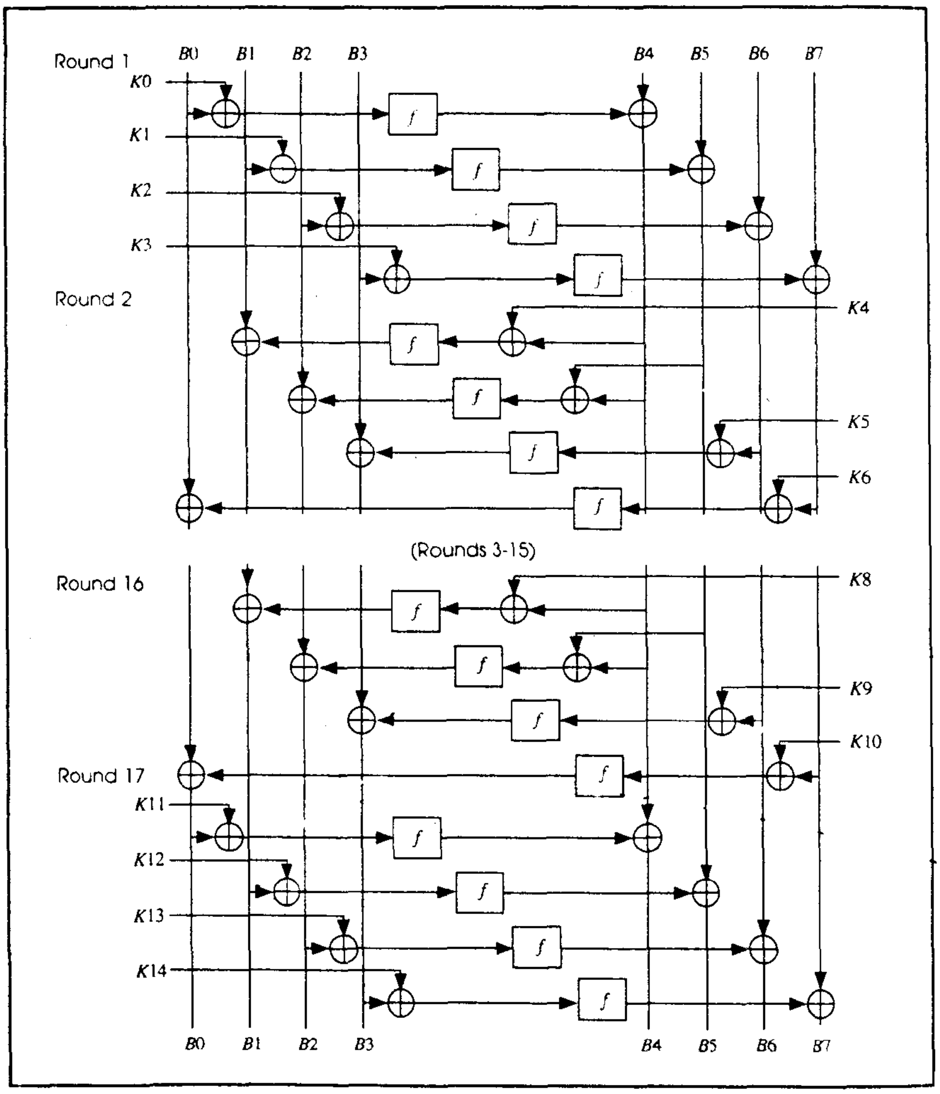


图 11.2 NewDES 加密算法

但密钥为120位。NewDES比DES简单，其中没有初始置换和最后置换，所有操作都是对整个字节处理的——NewDES不对某些位作读、写、或者置换处理。

NewDES 算法描述

明文块分成8个1字节长的子块：$B_0, B_1, \cdots, B_6, B_7$，各子块经过17轮处理，每一轮有8步。在每一步中，其中一个子块与密钥某些内容异或(一个例外)，通过函数f由另一字节替代，再与另一子块异或形成另一子块。密钥为120位，分成15个子块：$K_0, K_1, \cdots, K_{13}, K_{14}$。图11.2为NewDES加密算法。

函数f来源于"独立宣言"。

NewDES 密码分析

Scott证明了只需7轮之后，明文中的每一位均影响密文中的每一位，他还分析了函数f，没有发现什么明显的问题。C. Connell注意到NewDES具有同DES一样的互补密钥性质：如果 $E_K(P)=C$，则 $E_{K'}(P')=C'$，这把蛮力攻击所需的时间从 2^{120} 减少到 2^{119} 步。Biham注意到密钥中各个字节和各数据字节(B_5 除外)的任何改变都会导致另一个互补性质，这再度将蛮力攻击的需求减少到 2^{112} 步。

Biham的相关密钥密码分析攻击可用 2^{33} 个这样密钥的选择明文在 2^{48} 步内攻破NewDES。虽然该攻击相当费时且主要是理论上的，但它使NewDES远比老的DES脆弱。

11.4 FEAL-N 算法

FEAL-4是日本学者宫口庄司和清水明宫设计的，它采用64位的块和64位的密钥。FEAL-4类似于DES，但其中每一轮都比DES更强，并且迭代轮数比DES少，故效率更高。

FEAL 算法描述

图11.3是FEAL算法的部分描述，算法处理64位的数据块(明文或密文)。首先，数据块

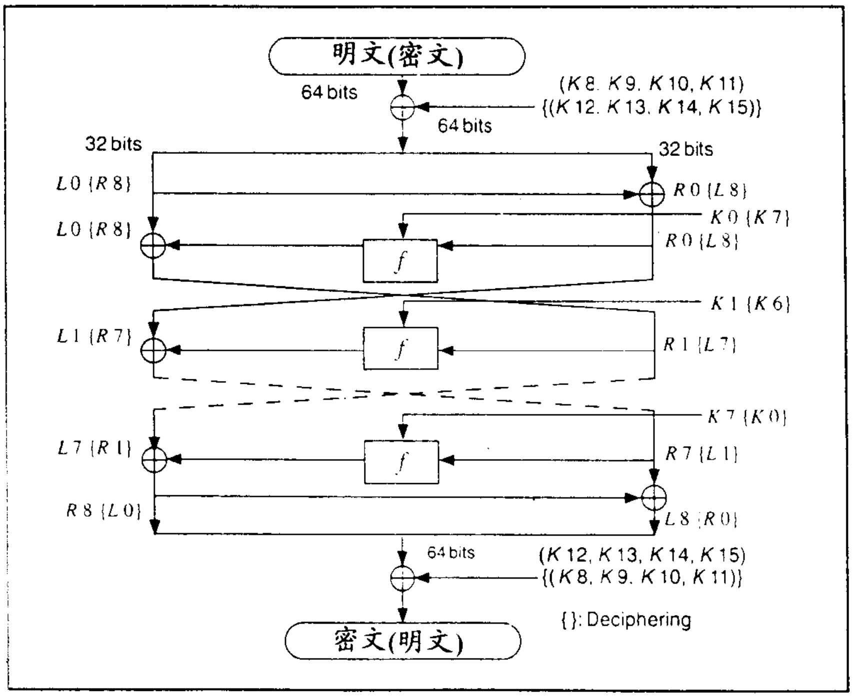


图 11.3 FEAL 的一次迭代

与密钥的某 64 位异或(64 位密钥如何构成将在后面讨论),接着将数据块分成左右二半,左半部分与右半部分异或形成新的右半部分,二部分经过 N 轮(最初为 4 轮)迭代处理。每一轮中右半部分与 16 位子钥结合(使用函数 f)并与左半部分异或形成新的右半部分,迭代之前的原右半部分作为新的左半部分。迭代 N 轮之后(左右二半在第 N 轮之后不得交换),左半部分再与右半部分异或形成新的右半部分,二部分再合并在一起形成 64 位,再与另外的某 64 位密钥异或得到最终的密文,算法结束。

函数 f 把 32 位数据和 16 位子钥混合在一起,首先,数据分成 2 个 8 位的块,各块相互异或和替代。图 11.4 是函数 f 的示意图,其中 S_0 和 S_1 函数定义如下:

$S_0(a,b) = ((a+b)$循环左移 2 位$) \bmod 256$

$S_1(a,b) = ((a+b+1)$循环左移 2 位$) \bmod 256$

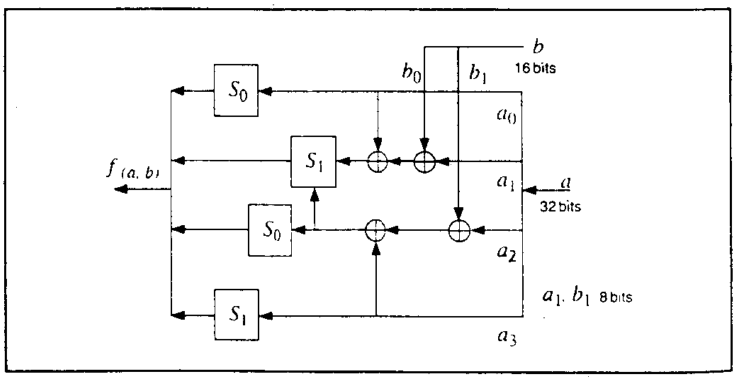


图 11.4 函数 f

加密和解密可以使用同一个算法,唯一的区别是密钥各部分使用的顺序相反。

图 11.5 是子密钥生成函数,首先,64 位密钥分成左右二半,二部分由函数 f_K 异或和处理。图 11.6 是函数 f_K 的示意图,32 位输入分成 4 个 8 位的块,S_0 和 S_1 如上定义,16 位的子密钥用于加密/解密算法。

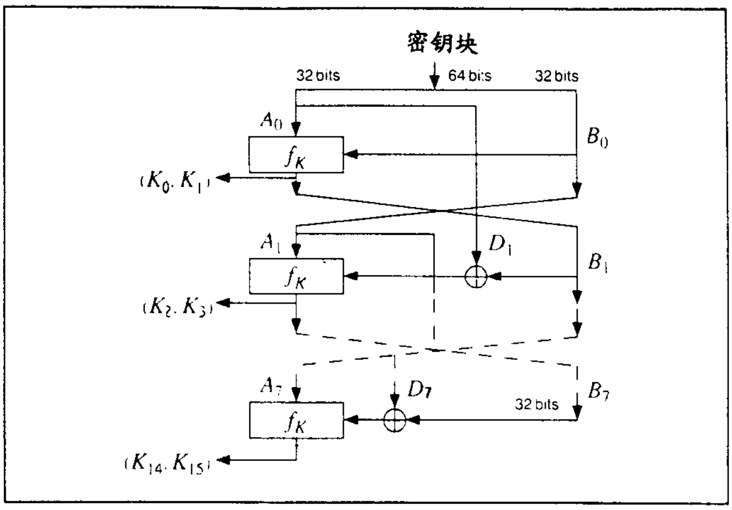


图 11.5 FEAL 的子密钥生成函数

用 10MHz 的 80286 微处理器上,FEAL-32 的汇编语言实现可以以 220kbit/s 的速度加密数据,FEAL-64 可以以 120kbit/s 的速度加密数据。

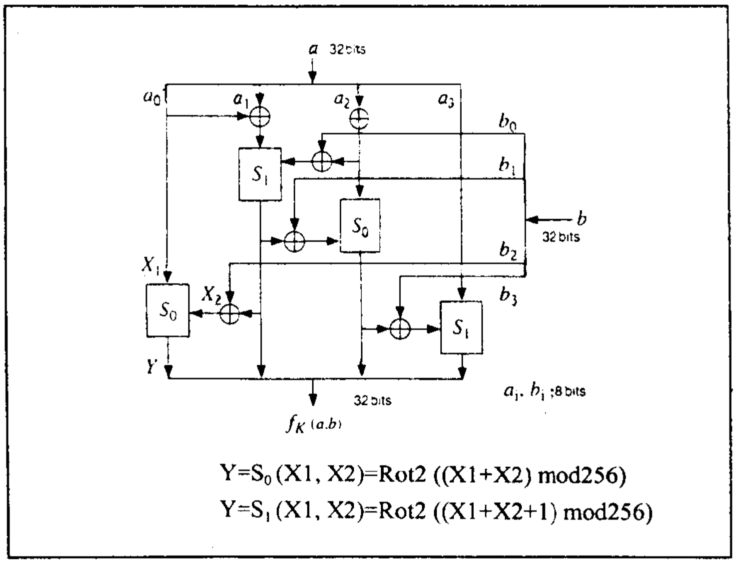


图 11.6 函数 fk

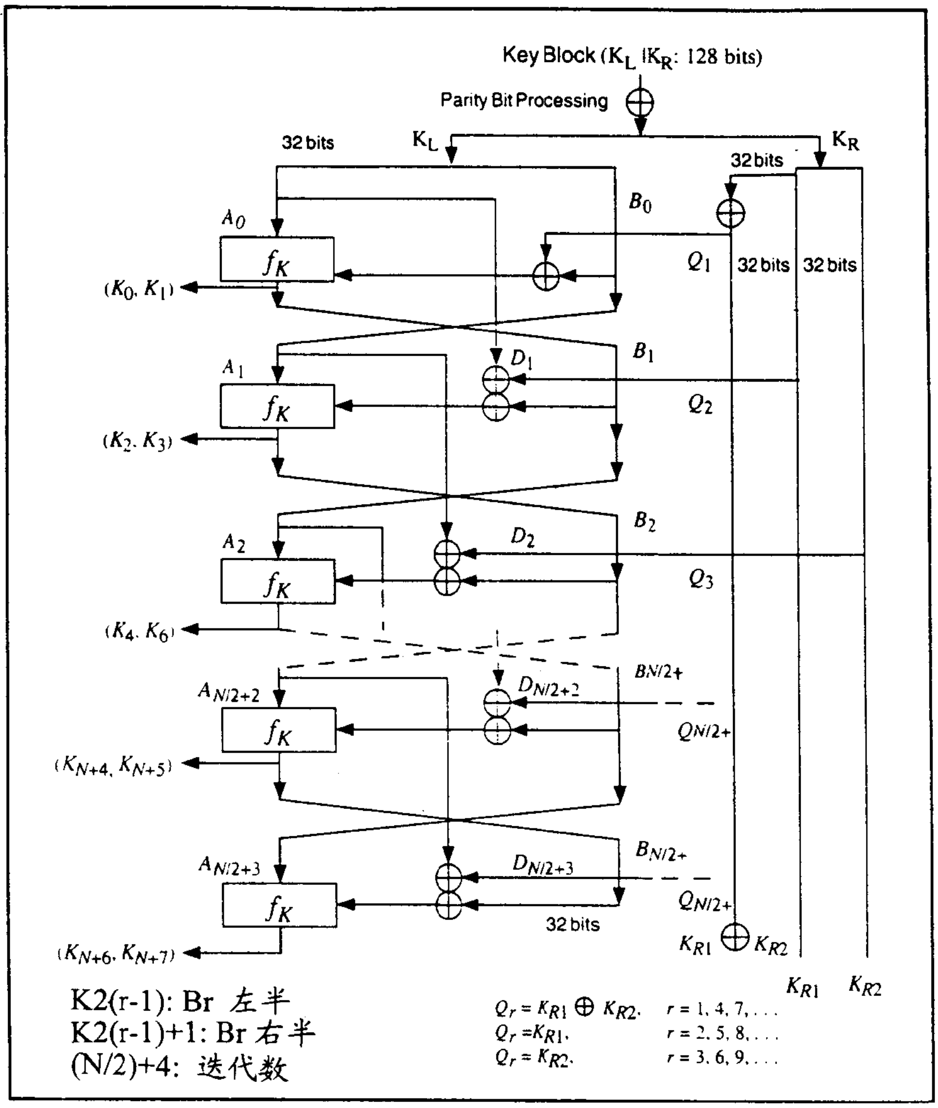


图 11.7 FEAL-NX 子密钥生成

FEAL 密码分析

4 轮迭代的 FEAL，称为 FEAL-4，有人成功地用选择明文攻击破译，后来 FEAL 的设计者又提出了 FEAL-8。Biham 和 Shamir 在 SECURICOM'89 会议上宣布对 FEAL-8 的一种成功的选择明文攻击方法，另一种对 FEAL-8 的选择明文攻击方法使设计者认输并定义了一种迭代轮数可变(当然大于 8)的 FEAL-N。

Biham 和 Shamir 把差分密码分析技术用于 FEAL-N，发现可以比穷举搜索快得多地攻破小于 32 轮的 FEAL-N：FEAL-16 需要 2^{28}个选择明文或 $2^{46.5}$个已知明文，FEAL-8 需要 2000 个选择明文或 $2^{17.5}$个已知明文，FEAL-4 只需 8 个精心挑选的选择明文就可攻破。

FEAL 的设计者还定义了 FEAL 的一种接受 128 位密钥的修正版：FEAL-NX(见图 11.7)，Biham 和 Shamir 证明了攻破 128 位密钥的 FEAL-NX 与 64 位密钥的 FEAL-N 一样容易。

攻击 FEAL-4 只需 1000 个已知明文，FEAL-8 只需 20,000 个已知明文。由 M. Matsui 和 A. Yamagishi 设计的最佳攻击方法可以用 5 个已知明文攻破 FEAL-4，用 100 个已知明文攻破 FEAL-6，用 2^{15}个已知明文攻破 FEAL-8，这是一种线性密码分析方法。

11.5 REDOC 算法

REDOC II 是由 M. Wood 设计的另一种块算法，其密钥为 20 字节(160 位)且对 80 位的数据块作处理。

REDOC II 中的所有操作----置换、替代和密钥异或都对字节处理，从而算法能以软件高效实现。REDOC II 还使用了可变函数表，而不像 DES 中置换和替代表都是固定的(当然对安全性作了优化)，REDOC II 采用各种密钥相关和明文相关的表。

REDOC II 设计中另一独特的特性是采用掩膜，从密钥表中得到一些数据用于某一轮中在给定函数内选择某些表，同时使用数据和掩膜来选择函数表。

REDOC II 的安全性

假如蛮力攻击是最有效的攻击方法，REDOC II 是非常安全的：找到密钥需执行 2^{160}次操作。

T. Cusick 分析了 REDOC II 的一轮，但他没能将攻击扩展到多轮迭代中，Biham 和 Shamir 成功地采用差分密码分析技术用 2300 个选择明文分析了 REDOC II 的一轮，但未能扩展到多轮中，他们也在 4 轮迭代之后获得了 3 个掩膜值，但也未能扩展攻击。

REDOC III

REDOC III 是由 M. Wood 设计的 REDOC II 的改进版，它处理 64 位的数据块，钥长可变并可长到 2560 字节(20,480 位)，算法只含有密钥字节与报文字节的异或运算，而没有置换和替代操作。

(1)用密钥建立一个由 256 个 10 字节密钥构成的密钥表。

(2)建立 2 个 10 字节的掩膜块，第 1 块是前 128 个 10 字节密钥的异或结果，第 2 块是后 128 个 10 字节密钥的异或结果。

(3)加密一个 10 字节的块：

(a)把数据块的首字节与第 1 个掩膜块的首字节异或。从密钥表中选择一个与第(1)步中得到的值(0～255 之间)对应的密钥，除首字节外，数据块的各个字节与选择的密钥中对应的字节异或。

(b)把数据块的第 2 字节与第 1 个掩膜块的第 2 字节异或。从密钥表中选择一个与第(1)步中得到的值(0～255 之间)对应的密钥,除第 2 字节外,数据块的各个字节与选择的密钥中对应的字节异或。

(c)对整个数据块继续处理(第 3～10 字节),直到每一字节在与相应的第 1 个掩膜块的值异或之后,都用于从密钥表中选择一个密钥,并且除了用于选择密钥的字节外,每一字节与密钥对应的字节异或。

(d)对第 2 个掩膜块重复(a)到(c)

该算法的执行速度是很快的,在 33MHz 的 80386 机上加密速度可达 2.75Mbit/s,Wood 估计在 64 位数据通路的 VLSI 流水线上,以 20MHz 时钟加密速度可超过 1.28Gbit/s。

REDOCIII 因其新近出现,仍难论及它的安全性,粗看起来简单而脆弱。

11.6 LOKI 算法

LOKI 于 1990 年首次由澳大利亚人提出是作为 DES 的可能替代品,它采用 64 位的数据块和 64 位的密钥。

LOKI 算法描述

LOKI 的机制类似于 DES。首先,数据块与密钥异或(不象 DES 的初始置换,这易于软件实现并有某些密码术优点),数据块接着非常类似于 DES 分成左右二半并经 16 轮迭代,每一轮中,右半部分先与一个子钥异或再经扩排列、S 盒替代和一个置换。总共有 4 个 S 盒,每个 S 盒由 12 位输入而产生 8 位输出。最后,右半部分与左半部分异或作为新的左半部分,而左半部分成了新的右半部分。16 轮迭代之后,数据块再次与密钥异或形成密文。

子钥从密钥生成:首先,64 位的密钥分成左右二半,每一轮中的子钥取左半部分,左半部分再循环左移 12 位并与右半部分交换。与 DES 一样,同一个算法既可用于加密又可用于解密,只要在如何使用子钥上作点修改即可。

LOKI 密码分析

Biham 和 Shamir 可以用差分密码分析技术攻破 11 轮以下的 LOKI,并且比蛮力攻击快。而且,其中一个 8 位互补特性,使蛮力攻击的复杂性降低了 256 倍。这样,16 轮以上的 LOKI 还是能抗差分分析的,对 64 位密钥蛮力攻击只需 2^{56}次加密。

Knudsen 提出了另一种差分密码分析的结果,证明了 14 轮以下的 LOKI 对差分分析是敏感的。另外,如果以交替的 S 盒实现 LOKI,则结果密文可能对差分分析是敏感的。

LOKI91

作为这些攻击的反应,LOKI 的设计者重新修正了算法,其结果为 LOKI91,而原 LOKI 更名为 LOKI89。

为使算法更能抗差分分析并去除密钥互补特性,在原设计上作了以下改变:

(1)子钥生成算法中二半部分每隔一轮交换而不是每轮交换。

(2)子钥生成算法中左半部分交替循环左移 12 位和 13 位。

(3)取消数据块在初始和最后时的与密钥异或操作。

(4)修改了 S 盒函数。

LOKI91 的大体结构如图 11.8 所示。

LOKI91 密码分析

L. Knudsen 试图破译 LOKI91,他没能对修正算法作差分分析,然而他发现了一种选择明

文攻击方法，使蛮力搜索的复杂性几乎降低了 4 倍，这个攻击利用了子钥生成中的一个弱点，并且当算法用作单向哈希函数时它也可能是弱点。

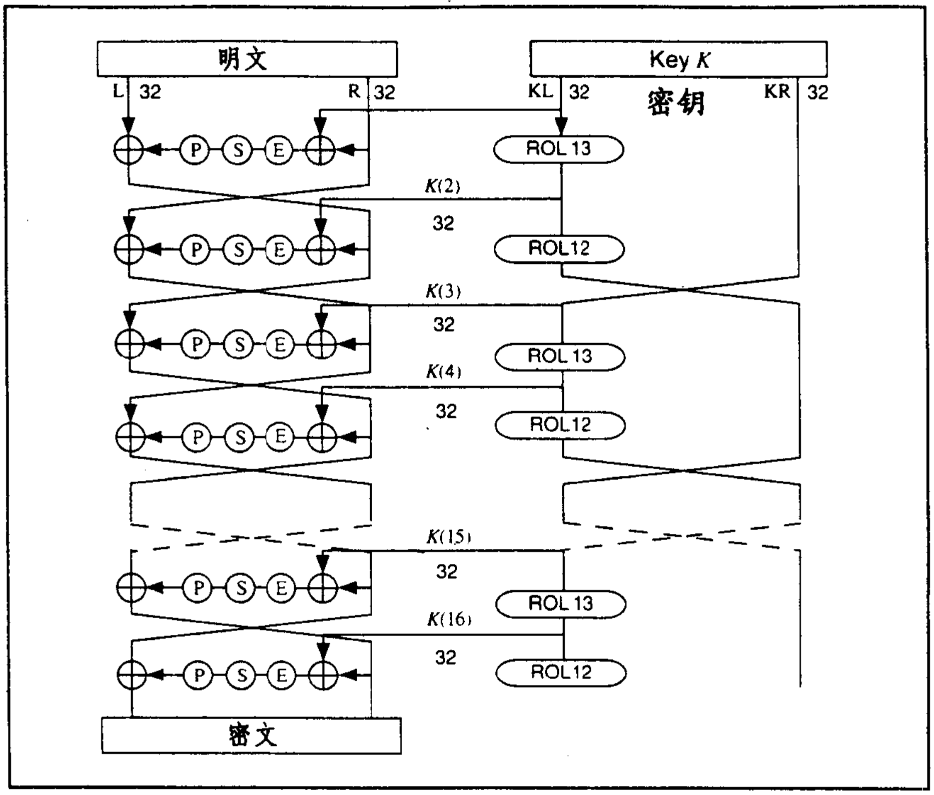


图 11.8　LOKI91 结构

Biham 的相关密钥密码分析法可用 2^{32} 个选择密钥的选择明文、或 2^{48} 个选择密钥的已知明文攻破 LOKI91，该攻击与算法的轮数无关。(Biham 还用 2^{17} 个选择明文或 2^{33} 个已知明文攻破了 LOKI89)。要使 LOKI91 抗这种攻击也是容易的，只需避免采用这种简单的密钥安排方法。

11.7　KHUFU 与 KHAFRE 算法

1990 年 R. Merkle 认识到 DES 快结束其生命期并提出了二个密码系统作为可能的替代品，当时其基本设计原则如下：

1. DES 的 56 位钥长太小。考虑到增长密钥的代价是微不足道的(计算机存储器既廉又丰)，应该增长密钥。

2. DES 中广泛使用置换操作，虽适合于硬件实现，但软件实现是很困难的。DES 的快速软件实现通过查表法实现置换，查表法可提供与置换同样的“扩散”特性并更为灵活。

3. DES 中 S 盒太小，每个盒只有 64 个 4 位项。既然存储器更大了，S 盒也应加大，而且应同时使用 8 个 S 盒。虽然这适合于硬件实现，但对软件似乎是个不合理的限制，应该使用更大并且是顺序的(而不是并行的)S 盒。

4. DES 中的初始和最后置换普遍认为在密码效果上没有意义，应取消。。所有 DES 的快速实现都预计算每轮中的密钥，因此没有理由不把该计算设计成更为复杂。

5. 不像 DES，S 盒的设计应该公开。

Khufu

Khufu 是 64 位的块密码算法，64 位明文块首先分成各为 32 位的左右二半 L 和 R，L 和 R

都与同一子钥异或，再经过类似于 DES 的几轮迭代。每一轮中，L 的最低字节用作 256 项 S 盒的输入，S 盒中每一项为 32 位。S 盒中选取的 32 位项与 R 异或，接着 L 循环移位 8 的倍数位，L 与 R 交换，一轮结束。S 盒本身也不是静止的，每 8 轮改变一次。最后一轮之后，L 和 R 与更多的子钥异或，并且合并在一起形成密文块。

虽然密钥的某些部分在算法开始和结尾处与加密块异或，密钥的主要目的是生成 S 盒，这些 S 盒是保密的且本质上是部分密钥。Merkle 的算法需要 512 位(64 字节)钥长，并由一个算法从密钥生成 S 盒。算法的轮数还未公开，Merkle 推测几乎所有应用场合都采用 16，24 或 32 轮(他把轮数限制为 8 的倍数)。

由于 Khufu 密钥相关且有保密的 S 盒，差分分析对它毫无结果。如果蛮力攻击是攻击 Khufu 的最好方法，它是非常难以攻破的：任何情况下，2^{512}的复杂度是不可想象的。

Khafre

Khafre 是由 Merkle 提出的二个密码系统中的另一个(Khufu 和 Khafre 是以二位埃及法老而取名的)，Khafre 的设计类似于 Khufu，只是 Khafre 是为没有预计算时间的应用场合而设计的。S 盒独立于密钥，Khafre 采用一组标准的 S 盒，并且密钥与加密块异或不仅在首轮之前和末轮之后，而且每 8 轮之后都执行一次。

Merkle 推测 Khafre 中钥长为 64 或 128 位，并且 Khafre 的加密轮数比 Khufu 多，再加上 Khafre 的每轮都比 Khufu 更复杂，使得 Khafre 要比 Khufu 慢。另一方面，Khafre 没有预计算，所以加密少量数据要比 Khufu 快一些。

1990 年，Biham 和 Shamir 用差分密码分析技术攻击 Khafre，他们以约 1500 个选择明文攻破 16 轮的 Khafre，用 PC 机运行了约 1 个小时。如果采用已知明文攻击需要约 2^{38}次加密，对于 24 轮的 Khafre，选择明文攻击需要 2^{53}次加密，已知明文需要 2^{59}次加密才能攻破。

11.8 RC2 和 RC4 算法

RC2 和 RC4 是由 Ron Rivest 为 RSA 数据安全公司(RSADI)而设计的可变钥长的加密算法，其中“RC”代表“Ron’s Code”。

RC2 是一种可变钥长的块密码算法，是以 DES 的替代品目的而设计的。RC2 的软件实现比 DES 快 3 倍，RC2 以 64 位的块加密数据，RC2 是一种“混合(mix)-打碎(mash)”方式的块密码算法，其中没有 S 盒，只有混合和打碎操作。

RC4 是一种可变钥长的流密码算法，比 DES 快 10 倍，RC2 和 RC4 的速度都与钥长无关。

RSA 数据安全公司声称他们的算法与 DES 一样安全(同等钥长情况下)，但他们拒绝公开算法，使人们对此产生怀疑。他们愿意给分析者提供算法的细节，RSA 数据安全公司的首席科学家 B. Kalish 宣布 RC2 能抗差分密码分析(没有 S 盒)，流算法 RC4 能抗差分分析。然而值得注意的是 RC2 和 RC4 都没能象 DES 那样经受得住 20 年激烈的密码分析。

假设算法是安全的，并且蛮力攻击是找到密钥最有效的方法，那么算法的安全性取决于钥长。如果采用长密钥，则该算法比 DES 安全，如果采用短密钥，则安全性比 DES 差。

11.9 IDEA 算法

由 X. Lai 和 J. Massey 设计的 IDEA 密码最初形式于 1990 年问世，当时称为 PES(Proposed Encryption Standard)，第二年，Biham 和 Shamir 提出了差分密码分析技术之后，设计者又增强了密码对这种攻击的免疫能力，称新算法为 IPES(Improved Proposed Encryption

Standard)。IPES 于 1992 年改名为 IDEA(International Data Encryption Algorithm),IDEA 也许是迄今为止可以公开获得的最好和最安全的块算法。

IDEA 概要

IDEA 是一种块算法,它处理 64 位的明文块,钥长为 128 位,同一算法既可用于加密又可用于解密。

IDEA 采用了混乱和扩散技术,其设计宗旨是"不同代数混合运算",有三种都易于硬件和软件实现的混合代数运算:

- 异或
- 模 2^{16}加法(忽略溢出)
- 模 $2^{16}+1$ 乘法(忽略溢出)

并且,所有这些运算(算法中也只有这些运算----没有置换运算)都对 16 位的子块操作,所以 IDEA 在 16 位处理器上更为有效。

IDEA 算法描述

图 11.9 为 IDEA 算法的结构。64 位的数据块分成 4 个各为 16 位的子块 x_1,x_2,x_3 和 x_4,这 4 个子块是算法首轮迭代的输入。算法共有 8 轮迭代,在每一轮中,4 个子块都与其中 1 个子块和 6 个 16 位的子钥异或、相加和相乘。每二轮之间,第 2 和第 3 子块交换。

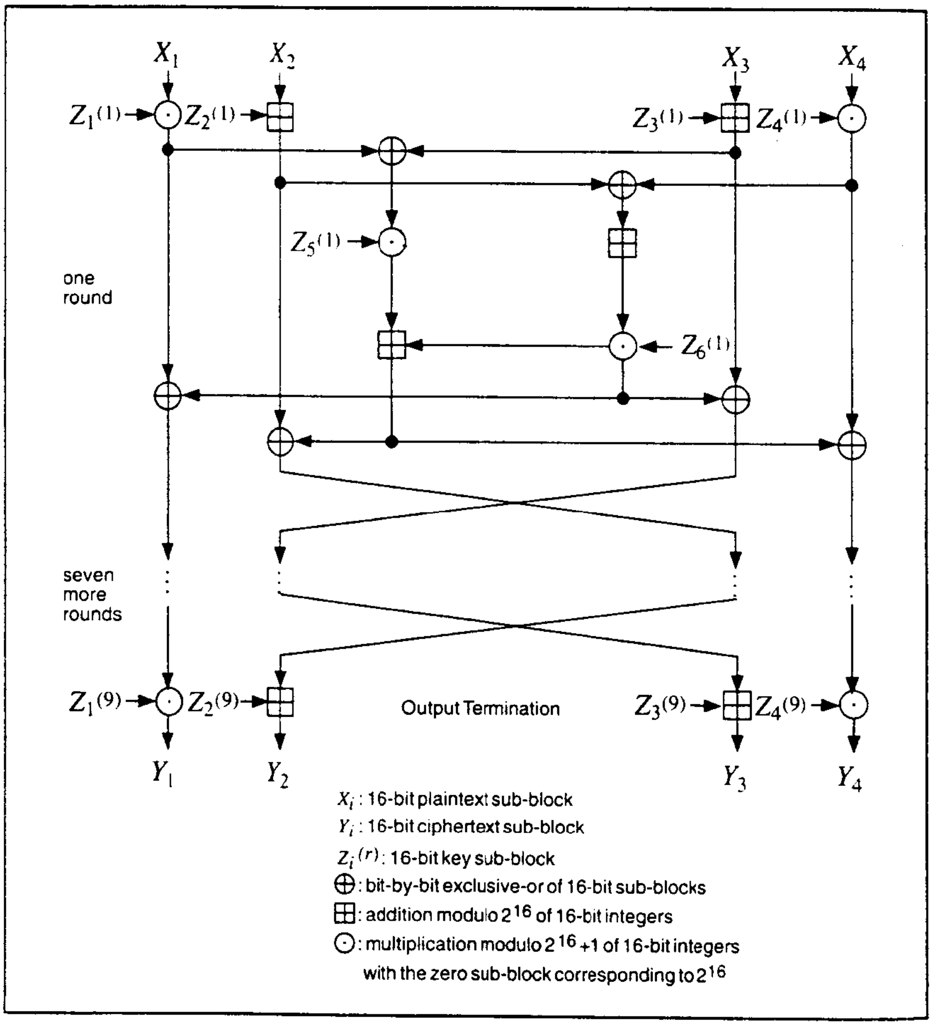


图 11.9 IDEA 结构

每一轮中的执行步骤如下：

(1)x_1 和第 1 子钥相乘

(2)x_2 和第 2 子钥相加

(3)x_3 和第 3 子钥相加

(4)x_4 和第 4 子钥相乘

(5)第(1)和第(3)步的结果异或

(6)第(2)和第(4)步的结果异或

(7)第(5)步的结果与第 5 子钥相乘

(8)第(6)和第(7)步的结果相加

(9)第(8)步的结果与第 6 子钥相乘

(10)第(7)和第(9)步的结果相加

(11)第(1)和第(9)步的结果异或

(12)第(3)和第(9)步的结果异或

(13)第(2)和第(10)步的结果异或

(14)第(4)和第(10)步的结果异或

首轮的输出是第(11)，(13)，(12)和(14)步结果的 4 个子块，交换中间二个子块(最后一轮除外)作为下一轮的输入。

第 8 轮之后，最后有一个输出转换过程：

(1)x_1 和第 1 子钥相乘

(2)x_2 和第 2 子钥相加

(3)x_3 和第 3 子钥相加

(4)x_4 和第 4 子钥相乘

最后，4 个子块合并在一起形成密文。

子钥生成也是容易的，算法共使用了 52 个子钥(8 轮中每轮 6 个和输出转换中的 4 个)。首先，128 位的密钥分成 8 个 16 位的子钥，用作算法的前 8 个子钥(第 1 轮中 6 个和第 2 轮中的前 2 个)，接着，密钥循环左移 25 位并再分裂成 8 个子钥，前 4 个用于第 2 轮，后 4 个用于第 3 轮，密钥再循环左移 25 位并分成下 8 个子钥，如此重复直到算法结束。

解密是完全一样的，除了子钥顺序相反并稍有不同，解密的子钥要么是加密子钥的加法逆，要么是其乘法逆(IDEA 中定义 0 的乘法逆为 0)。表 11.1 给出加密子钥与其相应的解密子钥。

表 11.1　IDEA 加密与解密子钥

轮	加密子钥						解密子钥					
1	$Z_1^{(1)}$	$Z_2^{(1)}$	$Z_3^{(1)}$	$Z_4^{(1)}$	$Z_5^{(1)}$	$Z_6^{(1)}$	$Z_1^{(9)-1}$	$-Z_2^{(9)}$	$-Z_3^{(9)}$	$Z_4^{(9)-1}$	$Z_5^{(8)}$	$Z_6^{(8)}$
2	$Z_1^{(2)}$	$Z_2^{(2)}$	$Z_3^{(2)}$	$Z_4^{(2)}$	$Z_5^{(2)}$	$Z_6^{(2)}$	$Z_1^{(8)-1}$	$-Z_2^{(8)}$	$-Z_3^{(8)}$	$Z_4^{(8)-1}$	$Z_5^{(7)}$	$Z_6^{(7)}$
3	$Z_1^{(3)}$	$Z_2^{(3)}$	$Z_3^{(3)}$	$Z_4^{(3)}$	$Z_5^{(3)}$	$Z_6^{(3)}$	$Z_1^{(7)-1}$	$-Z_2^{(7)}$	$-Z_3^{(7)}$	$Z_4^{(7)-1}$	$Z_5^{(6)}$	$Z_6^{(6)}$
4	$Z_1^{(4)}$	$Z_2^{(4)}$	$Z_3^{(4)}$	$Z_4^{(4)}$	$Z_5^{(4)}$	$Z_6^{(4)}$	$Z_1^{(6)-1}$	$-Z_2^{(6)}$	$-Z_3^{(6)}$	$Z_4^{(6)-1}$	$Z_5^{(5)}$	$Z_6^{(5)}$
5	$Z_1^{(5)}$	$Z_2^{(5)}$	$Z_3^{(5)}$	$Z_4^{(5)}$	$Z_5^{(5)}$	$Z_6^{(5)}$	$Z_1^{(5)-1}$	$-Z_2^{(5)}$	$-Z_3^{(5)}$	$Z_4^{(5)-1}$	$Z_5^{(4)}$	$Z_6^{(4)}$
6	$Z_1^{(6)}$	$Z_2^{(6)}$	$Z_3^{(6)}$	$Z_4^{(6)}$	$Z_5^{(6)}$	$Z_6^{(6)}$	$Z_1^{(4)-1}$	$-Z_2^{(4)}$	$-Z_3^{(4)}$	$Z_4^{(4)-1}$	$Z_5^{(3)}$	$Z_6^{(3)}$
7	$Z_1^{(7)}$	$Z_2^{(7)}$	$Z_3^{(7)}$	$Z_4^{(7)}$	$Z_5^{(7)}$	$Z_6^{(7)}$	$Z_1^{(3)-1}$	$-Z_2^{(3)}$	$-Z_3^{(3)}$	$Z_4^{(3)-1}$	$Z_5^{(2)}$	$Z_6^{(2)}$
8	$Z_1^{(8)}$	$Z_2^{(8)}$	$Z_3^{(8)}$	$Z_4^{(8)}$	$Z_5^{(8)}$	$Z_6^{(8)}$	$Z_1^{(2)-1}$	$-Z_2^{(2)}$	$-Z_3^{(2)}$	$Z_4^{(2)-1}$	$Z_5^{(1)}$	$Z_6^{(1)}$
输出转换	$Z_1^{(9)}$	$Z_2^{(9)}$	$Z_3^{(9)}$	$Z_4^{(9)}$			$Z_1^{(1)-1}$	$-Z_2^{(1)}$	$-Z_3^{(1)}$	$Z_4^{(1)-1}$		

IDEA 的速度

目前 IDEA 的软件实现几乎与 DES 一样快，在 33MHz 的 386 机上 IDEA 的加密速度为 880Kbps，VAX 9000 上的速度几乎是其 4 倍。

PES 的一种 VLSI 实现在 25MHz 时钟下加密速度为 55Mbit/S，另一种 ETH Zurich 开发的 VLSI 芯片在 25MHz 时钟下用 IDEA 算法加密数据的速度为 177Mbit/S。

IDEA 密码分析

IDEA 的钥长为 128 位，是 DES 的二倍多。假如蛮力攻击是最有效的攻击方法，找到密钥需执行 2^{128}(10^{38})次加密，每秒测试十亿个密钥的芯片也需执行 10^{13}年，10^{24}个这种芯片的阵列可在一天内找到密钥，但世界上没有足够的硅来制造这样的机器。

对 IDEA 作蛮力攻击还有另外一个困难，生成加密子钥要比生成解密子钥快得多。当加密或解密报文时，相对于加密或解密报文块的时间，生成子钥的时间是可以忽略的。但当对密文块作蛮力攻击时，每次试图作解密都得采用另一个密钥求得所有乘法逆元所需的时间会是蛮力攻击所需时间(或并行处理器数)的二或三倍。

然而，蛮力攻击也许并不是 IDEA 的最好攻击方法，这个算法太新以致于还没有差分密码分析的结果，设计者已尽力使算法对差分分析免疫(图 11.10 为原 DES 算法的结构，而后来的 IDEA 算法增强了抗差分分析能力，如此微妙的更改能有这么大的差异)。密码系统中的迭代

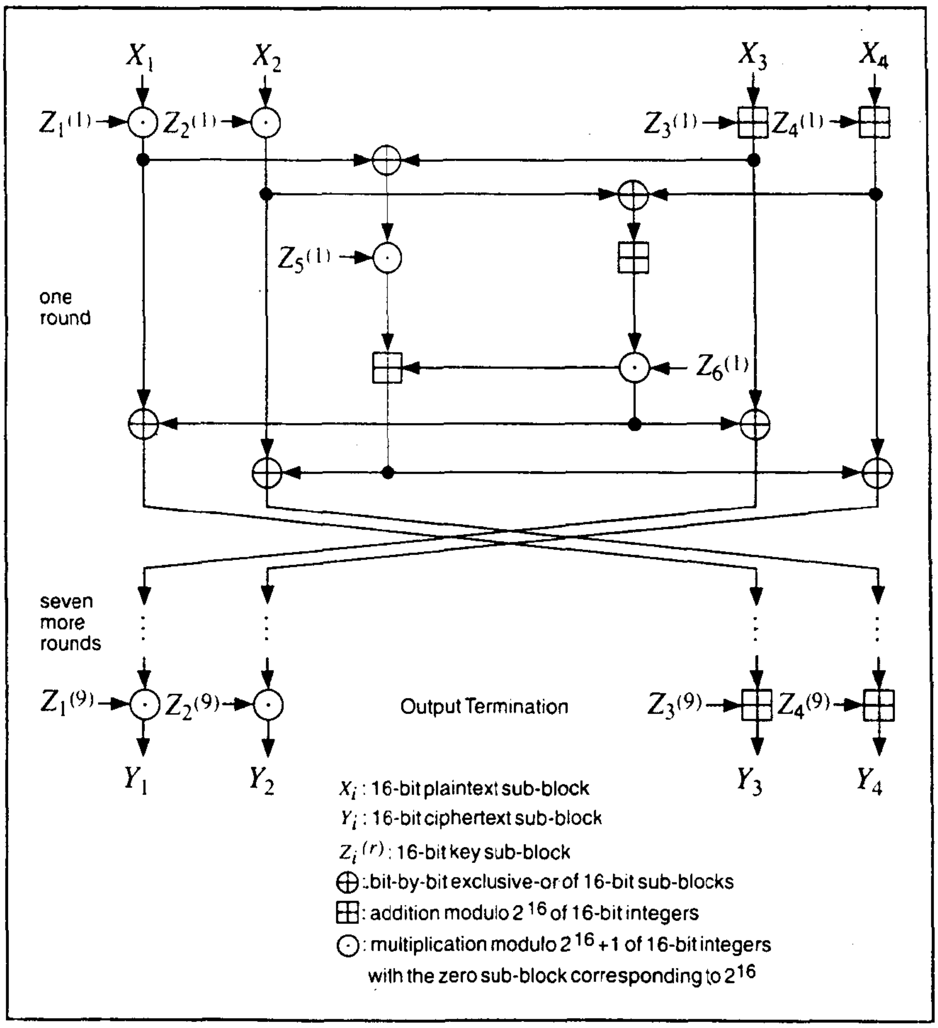


图 11.10 PES 结构

轮数越多则差分分析一般越困难，Lai 曾表示（他给出证据而不是证明）IDEA 只需 4 轮或 8 轮之后就对差分分析免疫，Biham 的相关密钥密码分析法也不能对 IDEA 生效。

最近有人"发现"了 IDEA 的一类弱钥，事实上，Lai 已描述了，它们在 DES 弱钥的意义上（加密函数是自逆的）不是弱钥，但如果使用了这些密钥，选择明文攻击可轻易标识出它们，在这个意义上它们是弱钥。例如，其中一个弱钥（十六进制）为：

0000 0000 0F00 0000 0000 000F FFFF F000

并且其中的"F"可以是任何数。

总之，偶尔生成其中这些弱钥的概率是很小的：2^{-96}，如果随机选择密钥几乎没有危险。

IDEA 的操作方式及变例

IDEA 可以以 8.1 中讨论的任何块密码方式工作，双重 IDEA 实现对 DES 中同样的中间会合攻击是敏感的，然而，由于 IDEA 的钥长为 DES 的二倍多，这种攻击是不可能的，它需要 64×2^{128} 位（10^{39} 字节）的存储空间。

如果担心并行攻击，采用三重 IDEA 实现：

$C = E_{K1}(D_{K2}(_{K1}(P)))$ 它对中间会合攻击是免疫的。

当然也可采用独立子钥实现 IDEA，算法总共需要 52 个 16 位的密钥，或者钥长共为 104 字节，这种变例的安全性会比 IDEA 更好。

一种自然的变例是把块大小加倍，算法也可以以 32 位的子块和 256 位的密钥运作，加密会更快，并且安全性会提高 2^{32} 倍。是否果真如此？算法的理论根据是 $2^{16}+1$ 是素数，而 $2^{32}+1$ 不是素数。也许修改后的算法可以运作，但其安全性特性非常不同，Lai 认为让其运作非常困难。

虽然 IDEA 比 DES 安全得多，但在现有应用场合中替代另一个算法并非易事。如果数据库和报文都设定接受 64 位的密钥，也许实现 128 位密钥的 IDEA 是不可能的，对于这些应用场合，可以把二个 64 位密钥连接在一起而得 128 位密钥，但会大大削弱 IDEA 的安全性。

IDEA 是一个新算法，在对它的密码分析方面还没有文章发表。至于它是一个群吗？是否一些密钥强于另一些等等都是有待研究的问题。

11.10 MMB 算法

MMB(Modular Multiplication-based Block cipher)算法抱怨 IDEA 采用了 64 位的加密块，MMB 基于与 IDEA 相同的基本理论——不同代数的混合运算。IDEA 是把 16 位异或、模 2^{16} 相加和模 $2^{16}+1$ 相乘混合，MMB 是把 32 位异或和模 $2^{32}-1$ 相乘混合，其结果算法有 128 位密钥和 128 位的数据块。

算法处理 32 位的明文子块 (x_0, x_1, x_2, x_3) 和 32 位的子钥 (k_0, k_1, k_2, k_3)，这使算法适合于现代 32 位处理器实现。非线性函数 f 由密钥异或交替使用 6 次。算法大致如下：

$x_i = x_i$ XOR k_i　　for $i=0$ to 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i$ XOR k_{i+1}　　for $i=0$ to 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i$ XOR k_{i+2}　　for $i=0$ to 3

$f(x_0, x_1, x_2, x_3)$

$x_i = x_i$ XOR k_i　　for $i=0$ to 3

$$f(x_0,x_1,x_2,x_3)$$
$$x_i = x_i \text{ XOR } k_{i+1} \qquad \text{for } i=0 \text{ to } 3$$
$$f(x_0,x_1,x_2,x_3)$$
$$x_i = x_i \text{ XOR } k_{i+2} \qquad \text{for } i=0 \text{ to } 3$$
$$f(x_0,x_1,x_2,x_3)$$

函数 f 有三步操作：

(1)$x_i = c_i * x_i$　　for i=0 to 3

(2)如果 x_0 的最低字节为 1，则 $x_0 = x_0$ XOR C

如果 x_0 的最低字节为 0，则 $x_3 = x_3$ XOR C

(3)$x_i = x_{i-1}$ XOR x_i XOR x_{i+1}　　for i=0 to 3

在第(1)步中的乘法操作是模 $2^{32}-1$ 的，在本算法中，如果第 2 操作数为 $2^{32}-1$，则结果为 $2^{32}-1$。算法中的各种常量如下(十六进制)：

$$C = 2AAAAAAA$$
$$C_0 = 025F1CDB$$
$$C_1 = 2C0$$
$$C_2 = 23C0$$
$$C_3 = 27C0$$

常量 C 是最低位为 0、无循环对称性的“最简单”的常量。常量 C_0 具有某些特征，常量 C_1，C_2 和 C_3 是 C_0 的移位结果以防止对称性攻击。

解密是加密的逆过程，第(2)和(3)步是其自身的逆，第(1)步中用 C_i^{-1} 代替 C，C_0^{-1} 的值为 0DAD4694。

MMB 的安全性

MMB 算法非常新，虽然 128 位密钥和 128 位块大小特征是所希望的，但评论算法的安全性还为时过早。Biham 认为 128 位的块大小事实上削弱了算法的安全性，并且认为 MMB 算法对差分密码分析不免疫，MMB 算法中没有弱钥。

11.11 CA1.1 算法

CA 是由 H. Gutowitz 设计的建立在细胞自动机(Cellular Automata)之上的块密码算法，它以 384 位的块加密明文并且密钥为 1088 位(实际上是二个密钥，一个为 1024 位，另一个为 64 位)，由于细胞自动机的特性，当大量并行集成电路实现时 CA 算法的效率最高。

CA1.1 同时使用了可逆和不可逆细胞自动机规则，在可逆规则下，网络的每一状态来自唯一的先前状态，而在不可逆规则下，每一状态有多个先前状态。在加密时，不可逆规则向后迭代，从某一状态向后迭代时，随机选取一个先前状态，这个处理可以重复多次，向后迭代将随机信息与报文信息混合。CA1.1 使用一种特殊的部分线性不可逆规则，使可以快速建立任何给定状态的随机先前状态。有的加密阶段也使用可逆规则，可逆规则(状态子块的简单并行置换)完全是非线性的。不可逆规则完全从密钥信息中导出，而可逆规则同时依赖于密钥信息和在加密阶段以不可逆规则插入的随机信息。

CA1.1 起某种认证机制的作用，这种认证机制保证只有由该系统以某密钥生成的密文才能用同一密钥解密。

CA1.1 围绕块链接结构而建立，即报文块的处理过程与加密时插入的随机信息流的处理

过程部分分离，随机信息把加密各阶段连接在一起，也可用于把一些序列块的加密过程链接在一起。链接中的信息在加密装置内生成，即使是系统的合法用户也是访问不到的，所以选择明文攻击是得不到这些信息的。

11.12 SKIPJACK 算法

Skipjack 是 NSA 为 Clipper 和 Capstone 芯片研制的加密算法，由于这个算法列为保密内容，除了下列内容外，没人对其知道多少：

- 它是一种对称算法
- 它的密钥为 80 位
- 它可用作 ECB，CFB，OFB 和 CBC 方式
- 每个加密和解密操作有 32 轮处理
- NSA 在 1985 年开始设计，在 1990 年结束评价

没人知道这个算法是否安全，但 NSA 计划用 Skipjack 算法加密 DMS(DefenseMessaging System)似乎表明该算法是安全的，然而，许多人还是持怀疑态度，也许 NSA 故意在算法中安放了陷门。NIST 曾声明有许多专家参与评估算法的性能。

11.13 使用单向哈希函数

Karn

由 P. Karn 设计并公开的这个算法，使可逆转的加密算法超出某种单向哈希函数，一种类似的思想由 Zheng 等人设计。

该算法处理 32 字节的明文和密文块，密钥可以是任意长，虽然对于某些单向哈希函数某种钥长会有效得多，例如 MD4 和 MD5 单向哈希函数，96 字节(768 位)的密钥最好。

加密时首先把明文分成各为 16 字节的二半 P_l 和 P_r，再把密钥也分成各为 48 字节的二半 K_l 和 K_r，连接 K_l 和 P_r 并用 MD4、MD5 或 Snefru 哈希散列，再把散列的结果与 P_r 异或生成密文的右半部分 C_r，连接 K_r 和 C_r 并用 MD4 和 MD5 哈希散列，再把散列的结果与 P_l 异或生成 C_l，最后合并 C_r 和 C_l 形成密文。

解密是其逆操作，连接 K_r 和 C_r，哈希散列，与 C_l 异或形成 P_l，连接 K_l 和 P_l，哈希散列，与 C_r 异或形成 P_r。

算法的整体结构与本节中许多块算法相同，由于算法的复杂性在于单向哈希函数，该算法只有 2 轮迭代，并且由于密钥只用作哈希函数的输入，所以不能用选择明文攻击找到密钥——当然假设单向哈希函数是安全的。

Kuby-Rackoff

Kuby 和 Rackoff 证明了 Karn 是不安全的，考虑 2 个单块的报文 AB 和 AC，如果密码分析者知道第 1 个报文的明文和密文，并且知道第 2 个报文的前一半明文，则他们就可以轻易地计算出整个第 2 个报文，这种已知明文攻击只在某种情况下有效，但也是一个主要的安全性问题。

Kuby 和 Rackoff 建议采用 4 轮加密算法来避免这个问题，4 轮的 Karn 算法如下：

(1)把密钥分成二半 K_l 和 K_r

(2)把明文块分成二半 L_0 和 R_0

(3)连接 K_l 和 L_0，并用 MD4 或 MD5 哈希散列，把哈希结果与 R_0 异或生成 R_1；

$$R_1 = R_0 \text{ XOR } H(K_l, L_0)$$

(4)连接 K_r 和 R_1，并用 MD4 或 MD5 哈希散列，把哈希结果与 L_0 异或生成 L_1：

$$L_1 = R_0 \text{ XOR } H(K_l, L_0)$$

(5)连接 K_l 和 L_1，并用 MD4 或 MD5 哈希散列，把哈希结果与 R_1 异或生成 R_2：

$$R_2 = R_1 \text{ XOR } H(K_l, L_1)$$

(6)连接 K_r 和 R_2，并用 MD4 或 MD5 哈希散列，把哈希结果与 L_1 异或生成 L_2：

$$L_2 = R_1 \text{ XOR } H(K_1, L_1)$$

(7)合并 L_2 和 R_2 形成密文

这种方法还是很新的，但至少与单向哈希函数一样安全，它可能比 DES 的 IDEA 都更安全，并且必然更快。

MDC（Message Digest Cipher）

由 P. Gutmann 设计的 MDC 是一种把单向哈希函数用作 CFB 方式的块密码的方法，这种密码的速度实际上与哈希函数一样快并且至少与哈希函数一样安全。

本书中的 MDC 实现采用 MD5 算法，虽然实际上可采用任何单向哈希函数（例如 MD2，MD5，Snefru，N—Hash，SHA 等），MD5 做的转换如下：

data[64] + hash[16]---> hash'[16]

data + 前面 MD5 ---> 当前 MD5

然而，不是接受 64 字节的数据，MDC 接受 64 字节的密钥，使得 MD5 依赖于密钥。确切的方法为：

(1)置 data[64]为 0，置 hash[16]为 IV

(2)对加密密钥执行密码算法

(3)置 data[64]为哈希散列后的密钥

通常第 2 步要迭代数次以减慢字典攻击的速度以及将加密密钥的随机性均衡地分布到整个 64 字节缓冲区中。

第 12 章　公钥算法(一)

12.1 背　　景

公钥密码系统(public-key cryptography)的概念由 W. Diffie、M. Hellman 和 R. Merkle 提出,由此这一领域内出现了一种新的见解:密钥可以成对出现,即一个加密密钥和一个解密密钥,并且二个密钥之间不能相互生成。Diffie 和 Hellman 首次在 1976 年的美国全国计算机会议(National Computer Conference)上提出这个概念,数月之后,他们的创举性论文"New Directions in Cryptography "刊登在"IEEE Transactions on Information Theory "上 。由于"Communication of the ACM"的出版速度较慢,Merkle 的文章直到 1978 年才刊登。

从 1976 年以来,提出了大量的公钥密码算法,其中许多是不安全的,还认为是安全的算法中许多又是不实用的,不是密钥大得不实用,就是密文比明文长得多。只有一些算法是既安全又实用的,这些算法大多建立在 9.2 节讨论的复杂性难题之中。

在这些既安全又实用的公钥算法中,有的只适合于密钥分发,有的只适合于加密,有的只适合于数字签名,只有 RSA 和 ElGamal 二个算法既适合于加密又适合于数字签名。所有这些算法的速度都很慢,加密和解密速度远远低于对称算法,往往慢到无法支持大量的数据加密。

公钥算法的安全性

密码分析者在着手分析公钥算法时,他们可以选择任何报文来加密,就是说给定 $C=E_K(P)$,密码分析者可猜测 P 的值并且易于检验猜测。如果可能的明文数小到可以穷举搜索,这是个严重的问题,但可以通过在报文中填充随机位串来解决,即同样的明文加密成不同的密文。

公钥算法是为抗选择明文攻击而设计的,其安全性并存于从公钥推断私钥的难度和从密文推断明文的难度,但是大多数公钥算法对选择密文攻击尤其敏感。

所谓选择用密文攻击(chosen-ciphertext attack)是密码分析者何时选择报文并用私钥将这些报文解密,他们的目的是推断密钥或解密经密钥加密报文的算法,即:

给定:$C_1, P_1=D_K(C_1), C_2, P_2=D_K(C_2), \cdots, C_i, P_i=D_K(C_i)$

其中分析者选择 $C_1, C_2, \cdots, C_i$

推断:K,或者从 $C_{i+1}=E_K(P_{i+1})$推断 P_{i+1}的算法

必须理解选择密文攻击的重要性,即使我们有一个可证明是安全的、并且采用貌似安全的协议的公钥密码系统,密码分析者还是可能找到明文。其内在的原因是人人都可使用公钥——这一公钥密码系统最有用特征的直接结果。

所以,要考虑系统整体而不仅仅是独立的各部分。好的公钥协议中各用户组不能解密由其它用户组生成的任何报文,例如身份证明协议。

12.2 DIFFIE-HELLMAN 算法

Diffie-Hellman 是第一个公钥算法,它以从有限域中计算离散对数的难度而获得安全性。

Diffie-Hellman 可用于密钥分发，但不能用于加密解密报文。

A 和 B 可使用这个算法生成私钥。首先，双方约定二个大整数 n 和 g，其中 $1<g<n$，这二个整数无需保密，A 和 B 可以通过不安全的信道约定，甚至可以是一组用户之间共同的整数。

接着，协议执行如下过程：

(1) A 随机选择一个大整数 x，并计算

$$X = g^x \bmod n$$

(2) B 随机选择一个大整数 y，并计算

$$Y = g^y \bmod n$$

(3) A 把 X 发送给 B，B 把 Y 发送给 A(注意 A 和 B 分别要将 x 和 y 保密)

(4) A 计算

$$k = Y^x \bmod n$$

(5) B 计算

$$k' = X^y \bmod n$$

k 和 k′都等于 $g^{xy} \bmod n$，因为窃听信道的人只知道 n，g，X 和 Y，他们无法计算出 k，除非他们能够计算离散对数而找到 x 和 y，他们无法解决这个问题。所以 k 是 A 和 B 各自计算出来的私钥。

g 和 n 的选取对系统的安全性有重大影响，模数 n 应该是素数，$(n-1)/2$ 也应为素数，并且 g 应是模 n 的本原根，最重要的是 n 应该很大，至少 512bit，1028bit 则更好。

推广的 Diffie-Hellman 算法

这个算法也可在有限域内执行。Shmuley 和 McCurley 研究了模数 n 为合数的算法变例，Miller 和 Koblitz 把这个算法推广到椭圆曲线，T. ElGamal 用其基本思想研制了一个加密和数字签名算法。

这个算法也在有限域 $GF(2^k)$ 内执行，由于计算速度快得多，有的实现采用这个方法。类似地，密码算法也快得多，因此要慎重选择一个足够大的域来保证安全性。

三方以上的 Diffie-Hellman 算法

Diffie-Hellman 密钥交换协议很容易推广到三人或更多人使用。这里，假设 A，B 和 C 一起生成一个私钥。

(1) A 随机选择一个大整数 x，并计算

$$X = g^x \bmod n$$

(2) B 随机选择一个大整数 y，并计算

$$Y = g^y \bmod n$$

(3) C 随机选择一个大整数 z，并计算

$$Z = g^z \bmod n$$

(4) A 把 X 发送给 B，B 把 Y 发送给 C，C 把 Z 发送给 A

(5) A 计算

$$Z' = Z^x \bmod n$$

(6) B 计算

$$X' = X^y \bmod n$$

(7) C 计算

$$Y' = Y^z \bmod n$$

(8) A 把 Z′发送给 B,B 把 X′发送给 C,C 把 Y′发送给 A

(9) A 计算

$$k = Y'^{x} \bmod n$$

(10) B 计算

$$k = Z'^{y} \bmod n$$

(11) C 计算

$$k = X'^{z} \bmod n$$

密钥 k 等于 $g^{xyz} \bmod n$,也没有窃听者能计算出 k。该协议很容易推广到四人或更多人,只需增加计算轮数即可。

12.3 背包算法

这是由 R. Merkle 和 M. Hallman 设计的第一个广义的公钥加密算法,它只能用于加密,后来 Shamir 把它改进成可用于数字签名。背包算法的安全性在于背包问题,一个 NP-完全问题,尽管后来发现背包算法不安全,但它说明了 NP-完全问题如何用于公钥密码术。

所谓的背包问题是:给定一堆重量各不相同的物品,是否可能选择一些物品装入背包,使得背包等于目标重量?

例如,各物品的重量为 1,5,6,11,14 和 20,塞满目标重量为 22 的背包是可能的:用重量为 5,6 和 11 的物品,目标重量为 24 是不可能的。一般来说,解决这个问题的时间是随物品数而指数增长。

通常可将该问题定义如下:给定一个值为 $M_1, M_2, \cdots, M_n$ 的集合以及一个目标和 S,计算下式中的各 b 值:

$$S = b_1M_1 + b_2M_2 + \cdots + b_nM_n$$

各个 b 值可以是 0 或者 1,值为 1 表示对应物品在背包中,值为 0 表示不在背包中。Merkle-Hallman 背包算法隐藏的思想是对一个报文进行编码,而其解就是一个背包问题。长度等于物品数的一块明文选择各项物品(明文中的各位对应于 b 值:1 表示该项物品选中,0 表示不选中),而密文就是结果和。图 12.1 给出一个明文用背包问题加密。

Plaintext:	1 1 1 0 0 1	0 1 0 1 1 0	0 0 0 0 0 0	0 1 1 0 0 0
Knapsack:	1 5 6 11 14 20	1 5 6 11 14 20	1 5 6 11 14 20	1 5 6 11 14 20
Ciphertext:	1+5+6+20=	5+11+14=	0=	5+6=
	32	30	0	11

图 12.1 背包加密算法

背包问题用于密码学的诀窍在于实际上有二种不同的背包问题,一是在线性时间内可解的,另一个只有在指数时间内可解,可以修改容易的背包来建立难背包。公钥是个难背包,可以用于加密报文但不能用于解密报文,私钥是容易背包,可以方便地用于解密报文,不知道私钥的人只能试图去求解难背包问题。

超递增背包

什么是容易背包问题?如果权重序列是个超递增序列(superincreasing sequence),则其背

包问题是容易求解的。超递增序列是指这样的序列：其中每一项都大于所有先前的各项之和，例如{1,3,6,13,27,52}是个超递增序列，而{1,3,4,9,15,25}不是。

超递增序列是易于求解的。将目标和与序列中最大的数相比，如果目标和小于该数，则该数不在背包中；如果目标和大于该数，则它在背包中，背包的目标和中减去该数并转向序列中的次大数，重复上述处理。如果最后背包目标和为0，则有解，否则无解。

例如，考虑背包目标和为70，权重序列为{2,3,6,13,27,52}。最大数52<70，所以52在背包中，70－52＝18；下一个数27>18，因此27不在背包中；下一个数13<18，故13在背包中，18－13＝5；下一个数6>5，6不在背包中；继续处理得到2和3都在背包中。如果这是一个Merkle-Hallman背包加密块，由值为70的密文产生的明文为110101。

非超递增背包没有快速的求解算法，确定哪些物品在背包中的唯一方法是有次序地测试各个可能解，直到碰巧遇到正确解。考虑必需启发式的最快算法的时间以物品数的指数增长，在权重序列中增加一项，求解时间就得延长一倍，这比求解超递增背包要难得多。

Merkle-Hallman算法就是基于这个特征，私钥是超递增背包问题的权重序列，公钥是具有相同解的一般背包问题的权重序列。Merkle和Hallman的贡献就是设计了一种将超递增背包问题转换为一般背包问题的技术，他们使用了模算术来实现。

从私钥生成公钥

如果不涉足数论，这个算法执行过程如下：为得到一个一般的背包序列，取一个超递增背包序列，例如{2,3,6,13,27,52}，并将所有值乘以一个数n并以m取模。模数m应大于序列中的最大值，如56。乘数n与序列中的所有数都没有公因子，如31。则一般的背包序列为：

$$2 * 31 \bmod 56 = 6$$
$$3 * 31 \bmod 56 = 37$$
$$6 * 31 \bmod 56 = 18$$
$$13 * 31 \bmod 56 = 11$$
$$27 * 31 \bmod 56 = 53$$
$$52 * 31 \bmod 56 = 44$$

难背包将为{6,37,18,11,53,44}。

超递增背包序列为私钥，一般背包序列为公钥。

加密

加密一个二进制报文，首先将报文分成长度等于背包序列中项数的各块，再将报文中的1表示选中对应项，0表示不选中，计算背包的总和。

例如，假设报文为011000110101101110，用上述背包加密的过程如下：

报文 = 011000 110101 101110

011000 对应于 37＋18 = 55

110101 对应于 6＋37＋11＋44 = 98

101110 对应于 6＋18＋11＋53 = 88

密文为55,98,88。

解密

报文的合法接收者知道原来的超递增背包及其用于将其转换成一般背包的n和m值，为解密报文，接收者必须首先计算出n^{-1}，使得$n * n^{-1} = 1 \bmod m$。把每个密文值乘以n^{-1}模m就可得相应的明文值。

本例中，超递增背包为{2,3,6,13,27,52}，m 为 56，n 为 31，密文报文为 55,98,88。这里 n^{-1}为 47，因此各密文值乘以 47 模 56：

$$55 * 47 \bmod 56 = 48 = 011000$$
$$98 * 47 \bmod 56 = 53 = 110101$$
$$88 * 47 \bmod 56 = 46 = 101110$$

恢复后的明文到 011000 110101 101110。

实际实现

对于只有 6 项的背包序列，即使不是超递增的也不是非常难以求解。真正的背包含有约 200 项，超递增背包中的每一项的值为 200 至 400 比特长，模数为 100 至 200 比特。算法的真正实现采用随机序列产生器来生成这些值。

对于这类背包，用蛮力求解是无济与事的，如果一台计算机每秒可测试 100 万个可能结果，测试所有可能的背包值也得需要 10^{46}年，即使 100 万台计算机并行工作也不能在世界末日到来之前求解。

背包算法的安全性

攻破背包密码系统的不是百万台计算机，而是二位密码编制人员。首先，Herlestam 注意到经常可以恢复某一位明文，后来，Shamir 证明了在某种环境里可以攻破该算法，但没有能攻破一般的 Merkle-Hallman 系统。最后，Shamir 和 Zippel 发现了一些转换过程中的缺陷，使得可以从一般的背包中重新构造超递增背包。

12.4 RSA 算法

RSA 算法是在提出 Merkle 的背包算法之后不久问世的第一个完整的公钥算法，它既可用于加密又可用于数字签名。在过去数年中提出的所有公钥算法中，RSA 算法是最易于理解和实现的，也是最流行的。这个算法的命名取自三位创始人：Ron Rivest，Adi Shamir 和 Leonard Adleman 姓名的首字母，自 1978 年提出后已经经受了多年广泛的密码分析，至今还未发现其中的严重缺陷。

RSA 的安全性在于大数因子分解的难度。公钥和私钥是二个大素数(100 到 200 位十进制位或甚至更大)的函数，从其中一个密钥和密文恢复明文等价于将二个素数的乘积因子分解。

为生成这二个密钥，选择二个大素数 p 和 q，计算其乘积：

$$n = p * q$$

再随机选取加密密钥 e，使得 e 与$(p-1)*(q-1)$互素，最后用欧几里德算法计算解密密钥 d，使得

$$e * d = 1 \pmod{(p-1)*(q-1)}$$

即

$$d = e^{-1} \pmod{(p-1)*(q-1)}$$

注意 d 和 e 也是互素的，数字 e 和 n 是公钥，d 为私钥，二个素数 p 和 q 不再用到，但不能泄露。

加密报文时，首先将 m 分成若干以数字表示的块，使得每块都有一个唯一的模 n 表示(用二进制数据，选择小于 n 的最大的 2 的幂)，即如果 p 和 q 都是 100 位十进制的素数，则 n 有近 200 位，并且每一报文块 m_i 也近 200 位。密文 c 由类似报文块大小的 c_i 构成。加密公式为：

$$c_i = m_i^e \pmod n$$

解密时取每一密文块 c_i 并计算：

$$m_i = c_i^d \pmod n$$

由于

$$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1}$$
$$= m_i * m_i^{k(p-1)(q-1)+1} = m_i * 1 = m_i$$
(所有都模 n)

上述公式恢复了报文。RSA 算法概括见表 12.1。

表 12.1　RSA 加密

公钥	
n	二个大素数 p 和 q 的乘积(p 和 q 要保密)
e	与 $(p-1)*(q-1)$ 互素
私钥	
	$d = e^{-1} (\text{mod } (p-1)*(q-1))$
加密	
	$c = m^e \pmod n$
解密	
	$m = c^d \pmod n$

报文也可以用 d 加密而用 e 解密。

例如，令 p=47，q=71，则

$$n = p * q = 3337$$
$$(p-1) * (q-1) = 46 * 70 = 3220$$

加密密钥 e 必须与 3220 没有公因子，随机选取 e 为 79，故：

$$d = 79^{-1} (\text{mod } 3220) = 1019$$

该数用推广的欧几里德算法计算。公开 e 和 n，保密 d，抛弃 p 和 q。

加密报文

$$m = 6882326879666683$$

首先将其分成小块，这里 3 位的小块比较好，这样报文将以如下的 6 块 m_i 加密：

$$m_1 = 688$$
$$m_2 = 232$$
$$m_3 = 687$$
$$m_4 = 966$$
$$m_5 = 668$$
$$m_6 = 63$$

第一块加密过程为：

$$688^{79} (\text{mod } 3337) = 1570 = c_1$$

对其它报文块执行同样的操作，得到如下密文：

$$c = 1570\ 2756\ 2714\ 2276\ 2423\ 158$$

解密时需要用解密密钥 1019 执行同样的指数运算：

$$1570^{1019} (\text{mod } 3337) = 688 = m_1$$

剩余报文以同样方式恢复。

RSA 算法的安全性

RSA 算法的安全性推测完全依赖于大数因子分解问题，具体技术实现上并不一定这样，数学上还没有证明从 c 和 e 计算 m 时需要对 n 作因子分解。可能会有分析 RSA 算法完全不同的方法，然而，如果这种新方法可以使分析者推断出 d，它也会用作大数因子分解的新方法。

已经证明有些 RSA 的变例与因子分解一样难，也证明了从 RSA 加密的密文中即使恢复

一位信息也与对整个报文解密一样难。

然而，因子分解是最明显的攻击方法，敌手用公钥 e 和模数 n，为找到解密密钥 d。因此 n 必须要足够大，有的 RSA 的硬件实现中 n 值采用 512 比特(152 位)，有的甚至采用 664 比特(200 位)。

假设根据 9.4 节中给出的因子分解的复杂性，因子分解一个 664 比特的数需要执行 10^{23} 步，假设一台计算机每秒可执行 100 万步，并且 100 万台计算机的网络做这项任务也需要 4000 年。

RSA 算法的硬件产品

目前已生产了许多执行 RSA 加密的各种芯片，表 12.2 给出部分现有的 RSA 芯片。

表 12.2 现有的 RSA 芯片

公司	时钟速度	比特率（每 512bit）	时钟周期（每 512bit 加密）	工艺	芯片比特数	晶体管数
Alpha Techn.	25MHz	13k	0.98M	2μm	1024	100,000
AT&T	15MHz	19k	0.4M	1.5μm	298	180,000
British Telcom.	10MHz	5.1k	1M	2.5μm	256	—
Business Sim.	5MHz	3.8k	0.67M	门阵列	32	—
Calmos Syst.	20MHz	28k	0.36M	2μm	593	95,000
CNET	25MHz	5.3k	2.3M	1μm	1024	100,000
Cryptech	14MHz	17k	0.4M	门阵列	120	33,000
Cylink	16MHz	6.8k	1.2M	1.5μm	1024	150,000
Pijnenburg	25MHz	50k	0.256M	1μm	1024	400,000
Plessy Crypto.	—	10.2k	—	—	512	—
Sandia	8MHz	10k	0.4M	2μm	272	86,000

RSA 算法的速度

最快的 RSA 大约要比 DES 慢 1000 倍，实现 512bit 模数的 RSA 的最快的 VLSI 吞吐量为 64kbit/s。

都用软件实现，RSA 大约比 DES 慢 100 倍，随着技术的改进，二者的速度之比也许会稍有改变，但 RSA 绝不会接近对称算法的速度，这就是为什么大多数实际系统都只用 RSA 来交换 DES 的密钥，而用 DES 加密数据的原因。

对 RSA 作选择密文攻击

事例 1：E 窃听 A 的通讯，并设法收集了 A 以公钥用 RSA 加密的密文 C，E 希望将 C 解密，即得到 p：

$$p = c^d \bmod n$$

为恢复 p，她首先选择一个小于 n 的随机数 r，并得到 A 的公钥 e，接着计算：

$$x = r^e \bmod n$$
$$y = x * e \bmod n$$
$$t = r^{-1} \bmod n$$

如果 $x = r^e \bmod n$，则 $r = x^d \bmod n$，所以 $t = x^{-d} \bmod n$。

E 再设法让 A 用其私钥签名 y(必须让 A 对报文签名，而不是作哈希运算)，由此解密了 y，A 发送给 E：

$$u = y^d \bmod n$$

E 计算：

$$t * u \bmod n = x^{-d} * y^{d} \bmod n = x^{-d} * x^{d} * c^{d} \bmod n$$
$$= e^{d} \bmod n = p$$

E 得到了 p。

事例 2：T 是位计算机公证人，如果 A 想要公证某文档，她把文档发送给 T，T 用 RSA 数字签名签署并送回 A（这里没有用到单向哈希函数，T 用他的私钥加密整个报文）。

M 想要 T 签署一个报文，但是也许因为时戳是假的，也许因为报文是从别的人发送过来，或者是别的什么原因，T 不签署，称此报文为 N。

首先，M 选择一个任意值 X 并计算 $Y = X^e$，由于 e 是 T 的公钥，为别人验证他的签名，e 必须公开，所以 M 可轻易地得到 e。他接着计算 M＝YN，把 M 发送给 T 签署，T 回送 $M^d \bmod n$。现在 M 计算 $(M^d \bmod n)\ X^{-1}$，它等于 $N^d \bmod n$，即 M 得到了经 T 签署后的报文 N。

事实上 M 可以采用多种方法完成同样的效果，其中所利用的弱点是指数运算保持输入的倍乘结构，即

$$(XM)^d = X^d M^d \bmod n$$

事例 3：E 想要 A 签署 M_3，她生成二个报文 M_1 和 M_2，使得

$$M_3 = M_1 * M_2 (\bmod n)$$

如果她让 A 签署了 M_1 和 M_2，它就可计算 M_3：

$$M_3^d (\bmod n) = M_1^d (\bmod n) * M_2^d (\bmod n)$$

从中引得的教训是：千万不要签署陌生人发送过来的任意文档，先要用单向哈希函数或者采用完全不同的签名算法。

对 RSA 作公共模数攻击

假如人人采用同一个 n，大家可以有不同的 e 和 d 值，但是 n 相同。最明显的问题是如果同一报文用二个不同的密钥（具有相同的模数）加密，并且这二个密钥互素，则无需解密密钥就可恢复明文。

设 P 为明文，二个加密密钥为 e_1 和 e_2，公共模数为 n，则二个密文为：

$$C_1 = P^{e1} \bmod n$$
$$C_2 = P^{e2} \bmod n$$

密码分析者知道 n，e_1 和 e_2，C_1 和 C_2，下面说明如何恢复 P。

由于 e_1 和 e_2 互素，欧几里德算法可以求得 r 和 s，使得

$$r * e_1 + s * e_2 = 1$$

假设 r 是负数（r 和 s 总归有一个为负数），则再次使用欧几里德算法计算 C_1^{-1}，则：

$$(C_1^{-1})^{-r} * C_2^{s} = P \bmod n$$

对这类系统还有二种更精巧的攻击方法，一种是用概率方法因子分解 n，另一种不对模数因子分解而用确定性算法计算某人的私钥。教训是在用户组之间不要使用相同的 n。

对 RSA 的低指数攻击

为了加快加密速度并易于执行，有人建议采用小值 e 作公钥来“改进”RSA，然而，这是不安全的。Hastad 介绍了一种对小加密密钥的 RSA 成功的攻击方法，当 e 只有 n 的 1/4 大小时，M. Wiener 的攻击方法可以找到 e，小的解密密钥 d 同样也是个严重的问题。千万记着应选择大数作 e 和 d。

Moore 基于这些成功的攻击列出了使用 RSA 时的一些限制：

· 对于给定的模数，加密/解密指数对的知识使攻击者能够对模数作因子分解。

· 对于给定的模数，加密/解密指数对的知识使攻击者能够无需因子分解 n 就可计算其它加密/解密对。

· 在通讯网络使用 RSA 的协议中不能使用公共的模数。

· 协议中选择的指数不能太小。

而且，有一个安全的密码算法是不够的，整个密码系统必须安全，密码协议也必须安全，如果任何一处有缺陷，整个系统就不安全。

标准

RSA 是世界上大多数国家的实际上的标准，国际标准组织，ISO/IEC 9796 指定 RSA 用于数字签名，法国银行社团也对 RSA 作了标准化，美国目前对公钥加密还没有标准，但可能会对数字签名作标准化。

12.5 POHLIG-HELLMAN 算法

Pohlig-Hellman 加密算法类似于 RSA，它也不是对称算法，加密和解密时采用不同的密钥，它不是公钥方案，加密和解密密钥都要保密。

类似于 RSA：

$$C = M^e \bmod n$$
$$M = C^d \bmod n$$

其中

$$ed = 1 \bmod \text{一个复杂数}$$

不像 RSA，n 不是根据二个大素数而定义的：它保留部分密钥，如果某人有 e 和 n，就可计算 d，不知道 e 和 d 的敌手将得计算：

$$e = \log_M C \bmod n$$

已知这是一个难问题。

12.6 RABIN 算法

Rabin 算法的安全性在于求解模一个合数的平方根的难度，这问题等价于因子分解。

首先选择二个素数 p 和 q，二者都模 4 同余于 3，这二个素数为私钥，n=pq 为公钥。

加密报文 M(M<n)时，只需计算

$$C = M^2 \bmod n$$

解密时同样容易，但是由于接收者知道 p 和 q，可以用中国余数定理求解这二个同余项，计算

$$M_1 = C^{(p+1)/4} \bmod n$$
$$M_2 = p - C^{(p+1)/4} \bmod n$$
$$M_3 = C^{(q+1)/4} \bmod n$$
$$M_4 = q - C^{(q+1)/4} \bmod n$$

M_1，M_2，M_3 和 M_4 四者之一等于 M，如果报文是英语文本，选择正确的 M 是容易的。另一方面，如果报文是随机位流(例如密钥生成)或数字签名，则无法确定哪个 M_i 是正确的。

所以，Rabin 没有把它作为一个公钥密码算法提出，然而，后来 ElGamal 改进了该思想用于他的公钥算法中。

Williams

Williams 重新定义了 Rabin 算法以消除这些缺点，他把 p 和 q 以如下方式选择：

$$p = -1 \pmod 4$$

$$q = -1 \pmod 4$$

并且

$$N = p * q$$

还有一个小整数 S，使得 J(S,N)=－1(J 为 Jacobi 符号)。N 和 S 公开，秘密密钥 k 为：

$$k = 1/2 * (1/4 * (p-1) * (q-1)+1)$$

加密报文 M 时，计算 c_1，使得 $J(M,N)=(-1)C_1$，再计算 $M'=S^{c1} * M \pmod n$。类似于 Rabin 算法，$C=M'^2 \pmod N$，并且 $c_2= M' \bmod 2$，最后的密文为三元组：

$$(C,c_1,c_2)$$

解密 C 时，接收者用下式计算 M″：

$$C_k = +/- M'' \pmod N$$

M″由 c_2 签署，最后

$$M = S^{-c1} * (-1)^{c1} * M'' \pmod N$$

Williams 后来又进一步改进了这个方案：不是把明文报文平方，而采用立方；大素数模 3 同余于 1，否则公钥和私钥就相同的；并且对每个加密只有唯一的解密。

这个算法相对于 RSA 有一个优点：可以证明它与因子分解一样安全，但它对选择密文攻击是完全不安全的。

12.7 FEIGE-FIAT-SHAMIR 算法

A. Fiat 和 A. Shamir 提出了一种认证和数字签名算法，U. Feige，Fiat 和 Shamir 把这个算法修改为身份的秘而不宣的知情证明算法，这是已知最好的身份的秘而不宣的知情证明算法。

简化的 Feige-Fiat-Shamir 认证算法

在公布私钥之前，仲裁人随机选择一个模数 n，是二个大素数的乘积。实际实现时 n 应至少 512 比特长并可能接近 1024 比特，一组证明人可以共用 n。

生成 P 的公钥和私钥时，可信赖的仲裁人选择一个数 v，v 为模 n 的平方余数，即选择 v，使得 $x^2=v \bmod n$ 有解并且模 n 的 v^{-1} 存在，这是公钥。接着计算满足 $s=\mathrm{sqrt}(1/v) \bmod n$ 的最小的 s，这是私钥。

认证协议如下：

(1) P 选择一个小于 n 的随机数 r，计算 $x=r^2 \bmod n$，并把 x 发送给 V。

(2) V 发送给 P 一个随机位。

(3) 如果 b=0，则 P 把 r 发送给 V；
　　如果 b=1，则 P 把 $y=r * s \bmod n$ 发送给 V。

(4) 如果 b=0，V 验证 $x=r^2 \bmod n$，证明 P 知道 sqrt(x)；
　　如果 b=1，V 验证 $x=y^2 * v \bmod n$，证明 P 知道 sqrt(x/v)。

这协议是单个认证，P 和 V 重复执行协议 t 次，直到 V 确信 P 知道 s，这是 cut-and-choise 协议，如果 P 不知道 s，如果 V 发送给 P 一个 0 或 1，她可选择 r 而欺骗 V，每次欺骗 V 的可能性为 50%，而欺骗 t 次的可能性为 $1/2^t$。

另一种攻击这个协议的方法是 V 冒充 P，他可用另一个验证者 V' 初始化协议，在第(1)步中，他可以再使用一个 P 用过的 r，而不是选择一个随机数 r，然而，V' 在第(2)步中选择 V 给 P 发送的同一个 b 值的可能性为 1/2，这样，欺骗 V' 的可能性为 50%，欺骗 t 次的可能性为 $1/2^t$。

这样，P 必须不能再使用一个用过的 r，否则，如果 V 在第(2) 步中发送给 P 另一个随机位，他就可以得到 P 的二种反应。他只要收集到其中的 t，他就可以冒充 P。

Feige-Fiat-Shamir 认证算法

Feige，Fiat 和 Shamir 说明了并行结构如何提高每轮的认证数目以及减少 P 和 V 的交互影响。

首先如上所述生成二个大素数的乘积 n，为生成 P 的公钥和私钥，首先选择 k 个不同的数：$v_1, v_2, \cdots, v_k$，其中 v_i 是模 n 的平方余数，即选取 v_i，使得 $x^2 = v_i \bmod n$ 有解，$v_1, v_2, \cdots, v_k$ 串是公钥。接着计算满足 $s_i = \mathrm{sqrt}(1/v_i) \bmod n$ 的最小的 s_i，$s_1, s_2, \cdots, s_k$ 串一私钥。

协议如下：

(1) P 选择一个小于 n 的随机数 r，计算 $x = r^2 \bmod n$，并把 x 发送给 V。

(2) V 发送给 P 一个 k 位的随机二进制位串 $b_1, b_2, \cdots, b_k$。

(3) P 计算 $y = r * (s_1^{b1} * s_2^{b2} * \cdots * s_k^{bk}) \bmod n$。(她根据 V 送给她的随机二进制位串把 s 值乘在一起，如果 V 的首位为 1，则把 s_1 乘进去，如果 V 的首位为 0，则把 s_1 不乘进去)，把 y 发送给 V。

(4) V 计算 $x = y^2 * (v_1^{b1} * v_2^{b2} * \cdots * v_k^{bk}) \bmod n$。(他根据随机二进制位串把 s 值乘在一起，如果首位为 1，则把 s_1 乘进去，如果首位为 0，则把 s_1 不乘进去)。

P 和 V 重复执行协议 t 次，直到 V 确信 P 知道 $s_1, s_2, \cdots, s_k$。

P 欺骗 V 的机会为 $1/2^{kt}$，其中 t 为重复执行协议的次数。

例子

这里以小数字为例说明协议的工作过程。

如果 n=35(二个素数为 5 和 7)，则可能的平方余数为：

1：$x^2 = 1 \bmod n$ 有解：x=1,6,29,34

4：$x^2 = 4 \bmod n$ 有解：x=2,12,23,33

9：$x^2 = 9 \bmod n$ 有解：x=3,17,18,32

11：$x^2 = 11 \bmod n$ 有解：x=9,16,19,26

14：$x^2 = 14 \bmod n$ 有解：x=7,28

15：$x^2 = 15 \bmod n$ 有解：x=15,20

16：$x^2 = 16 \bmod n$ 有解：x=4,11,24,31

21：$x^2 = 21 \bmod n$ 有解：x=14,21

25：$x^2 = 25 \bmod n$ 有解：x=5,30

29：$x^2 = 29 \bmod n$ 有解：x=8,13,22,27

30：$x^2 = 30 \bmod n$ 有解：x=10,25

逆元(模 35)及其平方根为：

v	v^{-1}	$S = \mathrm{sqrt}(v^{-1})$
1	1	1
4	9	3

9	4	2
11	16	4
16	11	9
29	29	8

注意 14,15,21,25 和 30 与 35 不互素,没有模 35 的逆元。

这样,P 取得由 $k=4$ 个值{4,11,16,29}构成的公钥,相应的私钥为{3,4, 9,8}。协议如下:

(1) P 选择一个随机数 $r=16$,计算 $16^2 \bmod 35=11$,并送给 V

(2) V 送给 P 一个随机二进制串{1,1,0,1}

(3) P 验证 $16*(3^1*4^1*9^0*8^1) \bmod 35 = 11$,并送给 B

(4) B 验证 $31^2*(4^1*11^1*16^0*29^1) \bmod 35 = 11$

P 和 V 重复执行协议,每次都用不同的随机数 r,直到 V 满意为止。

这样的小数字是没有安全性的,但当 n 为 512 比特或更长时,V 就不能获取有关 P 的私钥的内容。

增强措施

在协议中嵌入身份信息是可能的,假设 I 为表示 P 的身份信息(如姓名、住址、社会保险号、帽子尺寸等等)的二进制位,用一个单向哈希函数 H(x)计算 H(I,j),其中 j 为连接在 I 上的一个小的随机数。求一个 j 的集合,使得 H(I,j)为模 n 的平方余数,这些 j 成为 $v_1,v_2,\cdots,v_k$,于是 P 的公钥为 I 和 j 列表,她在协议的第(1) 步之前将 I 和 j 列表发送给 V,V 从 H(I,j)生成 $v_1,v_2,\cdots,v_k$。

于是,V 与 P 成功地结束协议之后,通过给她从 I 导出的 v_i 的平方根,他确信知道模数因子分解的人已经验证了 I 与 P 之间的联系。

设计者 Feige,Fiat 和 Shamir 还对下列实现作了注释:

- 对非完美的哈希函数,应该把 I 和一个长随机串 R 连接在一起来随机化 I,R 由仲裁人选择并与 I 一起交给 V。
- 在典型实现中,k 为 1 至 18 之间,k 值太大由于会减少轮数, 从而减少时间及通讯的复杂性。
- n 值应至少 512 比特。
- 如果所有用户都选择他们自己的 n 并公布在公钥文件中, 他们就用不到仲裁人了,然而,这种象 RSA 的变例使这个算法相当不方便。

Fiat-Shamir 签名算法

把这个身份认证算法变成签名算法基本上就是把 V 变成一个哈希函数。Fiat-Shamir 数字签名算法比 RSA 的主要好处是典型的签名只需模乘法的 1%到 4%,所以,它比 RSA 快得多。

与身份认证算法一样,选择 n 位二个大素数的乘积,生成公钥 $v_1,v_2,\cdots,v_k$, 以及私钥 $s_1,s_2,\cdots,s_k$,使得 $s_i=\mathrm{sqrt}(1/v_i) \bmod n$。

(1) A 选择小于 n 的 t 个随机整数 $r_1,r_2,\cdots,r_t$,并计算 $x_1,x_2,\cdots,x_t$,使得 $x_i=r_i^2 \bmod n$。

(2) A 把报文和 x 串的连接哈希散列,生成一个位串 $H(m,x_1,x_2,\cdots,x_t)$,把前 k_t 比特作为 b_{ij},其中 $i=1,2,\cdots,t,j=1,2,\cdots,t$

(3) A 计算 $y_1,y_2,\cdots,y_t$,其中

$$y_i = r_i * (s_1^{bi1} * s_2^{bi2} * \cdots * s_k^{bik}) \bmod n$$

(4) A 把 m,所有的 b_{ij}和 y_i 发送给 B,他已经有 A 的公钥 $v_1, v_2, \cdots, v_k$

(5) B 计算 $z_1, z_2, \cdots, z_t$,其中

$$z_i = y_i^2 * (v_1^{bi1} * v_2^{bi2} * \cdots * v_k^{bik}) \bmod n$$

注意 z_i 应等于 x.

(6) B 验证 $H(m, z_1, z_2, \cdots, z_t)$的前 kt 位是 A 送给他的 b_{ij}值

象身份认证算法,签名算法的安全性与 $1/2^{kt}$成比例,也与因子分解 n 的难度有关。Fait 和 Shamir 指出当因子分解 n 的复杂性远大于 2^{kt}时伪造签名最容易,并且,由于生日类型攻击,他们推荐 kt 应由 20 提高到至少 72,他们建议 k=9,t=8。

改进的 Fait-Shamir 签名算法

S. Micali 和 A. Shamir 改进了 Fait-Shamir 协议,他们选择 $v_1, v_2, \cdots, v_k$ 为前 k 个素数:

$$v_1 = 2,\ v_2 = 3,\ v_3 = 5, \cdots$$

这是公钥。

私钥 $s_1, s_2, \cdots, s_k$ 为随机的平方根,由下式决定:

$$s_i = \mathrm{sqrt}(v_i^{-1}) \bmod n$$

这里,每个人必须有不同的 n。这个修改使验证签名更容易,生成签名所需的时间和签名的安全性都没有影响。

第 13 章　公钥算法(二)

13.1　GUILLOU-QUISQUATER 算法

Fiat-Shamir 算法是第一个实用的身份秘而不宣的知情认证算法，它通过增加迭代轮数及每次迭代的认证数优化了交换中所需的计算量。对某些应用如智能卡来讲，它并不理想。与外部世界交换是很费时的，并且认证所需的存储，使卡中有限的资源更趋紧张。

Guillou 和 Quisquater 设计了一种更适合于这类应用场合的秘而不宣的知情身份认证算法，身份认证和被认证者之间的交换次数、每次交换中并行认证数绝对都是最小：每次证实时一个认证只有一次交换。对于同等的安全性，Guillou-Quisquater 算法所需的计算是 Fiat-Shamir 算法的三倍。

与 Fiat-Shamir 算法一样，Guillou-Quisquater 算法也可能作为数字签名算法，基于前述同样的原因，它可适用于智能卡。

Guillou-Quisquater 身份认证算法

设 P 要向 V 证明智能卡的身份，P 的身份由一些凭证组成：卡名、有效期、银行帐号以及其它任何应用许可证构成的数据串，这个位串称为 J(其实这些凭证可以是个长串，经哈希散列成 J，这点复杂化并不改动协议)，J 类似于公钥。由智能卡所有用户共享的其它公共信息包括指数 V 及模 n，n 为二个秘密素数的乘积，私钥 B 由 $JB^v=1 \pmod n$ 计算而得。

P 把她的凭证发送给 V，于是她要向 V 证明这些凭证是她的，为此，她要让 V 确信她知道 B。协议如下：

(1)P 在 1 至 n－1 之间选择一个随机整数 r，计算 $T=r^v \pmod n$ 并发送给 V。

(2)V 在 0 至 v－1 之间选择一个随机整数 d，并把 d 发送给 P。

(3)P 计算 $D=rB^d \pmod n$ 并发送给 V。

(4)V 计算 $T'=D^vJ^d \pmod n$。如果 $T=T' \pmod n$，则认证成功。

其数学原理为：

$$T'=D^vJ^d=(rB^d)^vJ^d=r^vB^{dv}J^d=r^v(JB^v)^d=r^v=T(\text{mod}n)$$

由于 B 满足

$$JB^v=1(\text{mod } n)$$

Guillou-Quisquater 签名算法

这个认证算法可转化为签名算法，也适合于智能卡实现。

公钥和私钥构成同上，协议如下：

(1)A 在 1 至 n－1 之间选择一个随机整数 r，计算 $T=r^v \pmod n$。

(2)A 计算 d＝H(M，T)，其中 M 为要签署的报文，H(x)为单向哈希函数。

(3)A 计算 $D=r_B{}^d \pmod n$，签名由报文 M、二个计算出来的值 d 和 D、以及凭证 J 构成。

(4)B 计算 $T'=D^vJ^d \pmod n$，并计算 $d'=H(M,T')$。如果 $d=d'$，则 A 必定知道 B，签名有

效。

多重签名

如果多人要在同一个文档上签名情况又会如何?虽然各人可独立签名,但下述签名方法更有效。本例中 A 和 B 签名同一文档,C 验证签名,实际上也适用于无论多少人参与的签名过程。同前面一样,A 和 B 有各自唯一的 J 和 B 值:J_A 和 J_B,B_A 和 B_B,n 和 v 值在系统中公用。

(1)A 在 1 至 n−1 之间选择一个随机整数 r_A,计算 $T_A=r_A^v(\text{mod } n)$。

(2)B 在 1 至 n−1 之间选择一个随机整数 r_B,计算 $T_B=r_B^v(\text{mod } n)$。

(3)A 和 B 计算 $T=T_AT_B(\text{mod } n)$。

(4)A 和 B 计算 d=H(M,T),其中 M 为要签署的报文,H(x)为单向哈希函数。

(5)A 计算 $D_A=r_AB_A^d(\text{mod } n)$。

(6)B 计算 $D_B=r_BB_B^d(\text{mod } n)$。

(7)A 和 B 计算 $D=D_AD_B(\text{mod } n)$。签名由报文 M、二个计算出来的值 d 和 D、以及二者的凭证 J_A 和 J_B 构成。

(8)C 计算 $J=J_AJ_B$。

(9)C 计算 $T'=D^vJ^d(\text{mod } n)$,并计算 $d'=H(M,T')$。如果 $d=d'$,则多重签名有效。

这个协议可推广到任意多少人,对于多人签名,在第(3)步中把各自的 T_i 值乘在一起,第(7)步中把所有各自的 D_i 值乘在一起,验证多重签名时,第(8)步中把所有签名者的 J_i 值乘在一起。这种多重签名技术在银行业中特别有用。

13.2 ONG-SCHNORR-SHAMIR 算法

Ong、Schnorr 和 Shamir 用多项式模 n 技术设计了一类签名算法。

选取一个大整数 n(无需知道 n 的因子分解),再选取一个与 n 互素的随机整数 k,计算 h:

$$h=-k^{-2}(\text{mod } n)=-(k^{-1})^2(\text{mod } n)$$

公钥为 h 和 n,私钥为 k。

签名报文 M 时,首先生成一个与 n 互素的随机整数 r,再计算:

$$S_1=1/2^*(M/r+r)(\text{mod } n)$$

$$S_2=k/2^*(M/r-r)(\text{mod } n)$$

S_1 和 S_2 对为签名。

验证签名时,只要验证下式是否成立:

$$M=S_1^2+h^*S_2^2(\text{mod } n)$$

这里叙述的算法是基于平方多项式的,当它首次提出时曾悬赏成功的密码破译,后来被 Pollard 证明了它的不安全性。于是设计者基于三次方多项式修改了算法,但也被 Pollard 攻破了,设计者又提出了四次方的算法,后来也被攻破了,从此设计者再也没有提出这类算法。

13.3 ELGAMAL 算法

ElGamal 算法是 Rabin 算法的一个变例,它既可用于数字签名又可用于加密,其安全性在于计算离散对数的难度。

为生成密钥对,首先选择一个素数 p 以及二个小于 p 的随机数 g 和 x,再计算

$$y=g^x(\text{mod } p)$$

公钥为 y、g 和 p,g 和 p 在一组用户之间可共享,私钥为 x。

ElGamal 签名算法

签名报文 M 时，首先选择一个与 p－1 互素的随机数 k，再计算

$$a = g^k \pmod p$$

并用推广的欧几里德算法求解下面方程中的 b：

$$M = xa + kb \pmod{p-1}$$

签名是 a 和 b，随机数 k 必须保密。

验证签名时，只要验证下式是否成立：

$$y^a a^b \pmod p = g^M \pmod p$$

算法概括如表 13.1 所示。

例如，选取 p＝11，g＝2，选取私钥 x＝8，计算

$$y = g^x \pmod p = 2^8 \pmod{11} = 3$$

公钥为 y＝3，g＝2 和 p＝11。

认证 M＝5 时，先选取一随机数 k＝9，验证 gcd(9,10)＝1。计算

$$a = g^k \pmod p = 2^9 \pmod{11} = 6$$

用欧几里德算法求解 b：

$$M = xa + kb \pmod{p-1}$$

$$5 = 6 * 8 + 9 * b \pmod{10}$$

解答为 b＝3，签名为 a＝6 和 b＝3。

验证签名时，验证：

$$y^a a^b \pmod p = g^M \pmod p$$

$$3^6 6^3 \pmod{11} = 2^5 \pmod{11}$$

表 13.1 ElGamal 签名算法

公钥	
p	素数(可在一组用户之间共享)
g	<p(可在一组用户之间共享)
$y = g^x \pmod p$	
私钥	
x<p	
签名	
k	选择一个与 p－1 互素的随机数
a(签名)$= g^k \pmod p$	
b(签名)满足 $M = xa + kb \pmod{p-1}$	
验证	
如果 $y^a a^b \pmod p = g^M \pmod p$ 成立，则签名有效	

T. Beth 设计了一种适合于身份证明的 ElGamal 算法的变例。

ElGamal 加密算法

加密报文 M 时，首先选取一个与 p－1 互素的随机数 k，再计算：

$$a = g^k \bmod p$$

$$b = y^k M \bmod p$$

a 和 b 对为密文，注意密文为明文的二倍大小。

解密 a 和 b 时，计算

$$b/a^x \pmod p$$

由于 $a^x = g^{kx} \bmod p$，并且 $b/a^x = y^k M/a^x = y^{kx} M/g^{kx} = M \bmod p$，所以这个算法正确(见表 13.2)。

表 13.2 ElGamal 加密算法

公钥	
p	素数(可在一组用户之间共享)
g	<p(可在一组用户之间共享)
$y = g^x \pmod p$	
私钥	
x<p	
加密	
k	选择一个与 p－1 互素的随机数
a(密文)$= g^k \pmod p$	
b(密文)$= y^k M \bmod p$	
解密	
M(明文)$= b/a^x \pmod p$	

13.4 SCHNORR 算法

Schnorr 的认证和签名算法，组合了 ElGamal、Fiat-Shamir、D. Chaum，Jan-Hendrik Evertse，以及 J. Graaf 的交互协议的思想，其安全性在于计算离散对数的难度。

为生成密钥对，首先选取二个素数 p 和 q，使得 q 是 p－1 的质因子，再选择满足 $a^q=1$ mod p 的不为 1 的 a，所有这些数在一组用户之间可以公用并可自由公开。

为生成独特的公钥/私钥对，选择一个小于 q 的随机数 s，这是私钥。再计算 $v=a^{-s}$mod p，这是公钥。

认证协议

(1)P 选择一个小于 q 的随机数 r，并计算 $x=a^r$ mod p，这是预处理阶段。

(2)V 送给 P 一个在 0 至 2^t-1 之间的随机数 e。

(3)P 计算 y=r+se mod q 并把 y 送给 V。

(4)V 验证 $x=a^y v^e$ mod p。

安全性基于参数 t，攻破算法的难度约为 2^t。Schnorr 推荐 p 约为 512 比特，q 约为 140 比特，与这种安全性相称的 t 值为 72，Schnorr 认为是足够安全的。

数字签名协议

公钥/私钥对同上，但增加一个单向哈希函数 H(M)。

(1)A 选取一个小于 q 的随机数 r，并计算 $x=a^r$ mod p，这是预处理阶段。

(2)A 把报文 m 和 x 连接，并把结果作哈希散列运算：

$$e=H(m,x)$$

A 计算 y=r+se mod q，签名为 e 和 y，把 e 和 y 送给 B。

(4)B 计算 $z=a^y v^e$mod p，并验证报文 m 和 z 的连接哈希散列到 e：

$$e=H(m,z)$$

如果成立，则签名有效。

Schnorr 列举了这个算法的一些新奇特征：

·生成签名的大多数计算可在预处理阶段完成，而独立于要签名的报文。因此，可在空闲时间内完成而不影响签名速度。

·对于同等的安全性，Schnorr 的签名长度比 RSA 短。例如，用 140 比特的 q，签名只有 212 比特，还不到 RSA 签名的一半。Schnorr 的签名也比 ElGamal 短得多。

E. Brickell 和 K. McCurley 修改了这个算法，增强了安全性。

13.5 数字签名算法(DSA)

1991 年 8 月，NIST 提出了新的数字签名标准(DSS)和采用的数字签名算法(DSA)，其声明如下：

“提请 DSS(数字签名标准)列为联邦信息处理标准(FIPS)。DSS 适用于联邦数字签名之用，它指定了一个公钥数字签名算法(DSA)，用公钥验证接受数据的完整性及发送者的身份，也可由第三方用于确定签名及数据的真实性。”

声明的反应

NIST 的声明引起强烈的批评与诉讼，RSA 数据安全公司带头反对 DSS，他们只需要 RSA，而不是任何别的算法作为标准。RSADSI 在通过 RSA 算法上耗费了巨资，而一个无专利税的数字签名标准势必直接影响他们的基础。

许多接纳了 RSA 算法的大软件公司也出面反对 DSS，诸如 IBM，Apple，Novell，Lotus，Northern Telecom，Microsoft，DEC 以及 Sun 等公司都耗费了巨资实现 RSA 算法，他们不希

望失去投资。

截止 1992 年 2 月 28 日的第一个评论期结束,NIST 共收到了 109 个评论,其中反对 DSA 的批评主要如下:

(1)DSA 不能用于加密和密钥分发

(2)DSA 由 NSA 设计,算法中可能会有陷门

(3)DSA 比 RSA 慢

(4)RSA 是事实上的标准

(5)DSA 中的选择处理不公开,这给分析提供了充足的时间

(6)DSA 可能侵犯其它专利

(7)密钥太短

DSA 算法描述

DSA 是 Schnorr 和 ElGamal 签名算法的变例,算法中使用了下列参数:

p=一个 2^L 比特的素数,其中 L 为 512 至 1024 之间,并是 64 的倍数(在原标准中,p 的大小恒定为 512 比特,这是许多批评的攻击点,后来 NIST 作了修改)。

q=p−1 的 160 比特的质因子

$g=h^{(p-1)/q}$,其中 h 为满足 $h^{(p-1)/q} \bmod p>1$ 的小于 p−1 的任何数

x=小于 q 的数

$y=g^x \bmod p$

另外,算法还使用了一个单向哈希函数 H(x)。对于 DSS,这是安全哈希算法(Secure Hash Algorithm)。

前三个参数 p、q 和 g 是公开的,网络上各用户可共用。私钥为 x,公钥为 y。

签名报文 m 的过程如下:

(1)A 生成一个小于 q 的随机数 k。

(2)A 生成:

$$r=(g^k \bmod p) \bmod q$$

$$s=(k^{-1}(H(m)+xr)) \bmod q$$

参数 r 和 s 是她的签名,并送给 B。

(3)B 计算下列式子验证签名:

$$w=s^{-1} \bmod q$$

$$u1=(H(m)*w) \bmod q$$

$$u2=(r*w) \bmod q$$

$$v=((g^{u1}*g^{u2}) \bmod p) \bmod q$$

如果 v=r,则签名有效。

表 13.3 DSA 签名算法

公钥	
p	512 至 1024 比特的素数(可在一组用户之间共享)
q	p−1 的 160 比特的因子(可在一组用户之间共享)
g	$h^{(p-1)/q}$,其中 h<p−1 并且 $h^{(p-1)/q} \bmod p>1$ (可在一组用户之间共享)
$y=g^x(\bmod p)$(160 比特的数)	
私钥	
x<q(160 比特的数)	
签名	
k	选择一个小于 q 的随机数
$r(签名)=g^k(\bmod p) \bmod q$	
$s(签名)=(k^{-1}(H(m)+xr)) \bmod q$	
验证	
$w=s^{-1} \bmod q$	
$u1=(H(m)*w) \bmod q$	
$u2=(r*w) \bmod q$	
$v=((g^{u1}*g^{u2}) \bmod p) \bmod q$	
如果 v=r,则签名有效	

预计算

DSA 真正实现时可通过预计算来加快速度。由于 r 值与报文无关,因此可以生成一串随机的 k 值并分别预计算各个 r 值,也可对各个 k 值预计算 k^{-1},当报文到来时,就可对给定的 r 和 k^{-1} 计算 s。

预计算处理极大地提高了 DSA 的速度,表 13.4 是智能卡实现中 DSA 和 RSA 计算时间比较。

表 13.4 DSA 和 RSA 计算时间比较

	DSA	RSA	公共 p,q 和 g 的 DSA
全程计算	Off Card(P)	N/A	Off Card(P)
密钥分发	14s	Off Card(S)	4s
预计算	14s	N/A	4s
签名	0.03s	15s	0.03s
验证	16s	1.5s	10s
	1～5s Off Card(P)		1～3s Off Card(P)

注：· Off Card 计算在 33MHz 的 80386 PC 上执行

· (P)表示公开参数离卡

· (S)表示秘密参数离卡

· 二个算法都采用 512 比特的模数

DSA 素数生成

Lenstra 和 Haber 指出有些模数易于攻破，如果使用了这些“敏感的”模数，就可容易地伪造他们的签名。但这并不是严重的问题，因为第一，这种模数是很容易检测出来的，第二，这种模数非常少，随机选中这种模数的机会是微不足道的，事实上比用概率素数生成程序偶尔生成一个合数的机会还要小。

NIST 推荐了一种生成二个素数 p 和 q 的方法，用 q 除 p－1，其中素数 p 为 512 至 1024 比特并是 64 比特的倍数，素数 q 为 160 比特。令 $L-1=n*160+b$，其中 L 为 p 的长度，n 和 b 为二个数。

(1)选择一个至少 160 比特的任意序列，称之为 S，设 g 为 S 的比特数

(2)计算 $U=SHA(S) XOR\ SHA((S+1) \bmod 2^g)$

(3)把 U 的最高位和最低位设置为 1 而得到 q

(4)检验 q 是否为素数

(5)如果 q 不是素数，回到(1)

(6)设 C=0 和 N=2

(7)对于 k=0,1,…,n，设 $V_k=SHA((S+N+k) \bmod 2^g)$

(8)设

$$W=V_0+V_1*2^{160}+\cdots+V_{n-1}*2^{(n-1)*160}+(V_n \bmod 2^b)\times 2^{n*160}$$

$X=W+2^{L-1}$ 注意 X 为 2^L 比特的数

(9)设 $p=X-((X \bmod 2_q)-1)$，注意 p 模 2_q 同余于 1

(10)如果 $p<2^{L-1}$，转(13)

(11)检验 p 是否为素数

(12)如果 p 为素数，转(15)

(13)令 C=C+1 和 N=N+1

(14)如果 C=4096，则转(1)，否则转(7)

(15)保存用于生成 p 和 q 的 S 值和 C 值

这一方法的关键是生成 p 和 q 的过程是公开的，由于人们担心的是 DSA 易受攻击的“敏感的”p 和 q 值，为实用目的，该算法防止了这种情况。如果直接传送过来 p 和 q 值，你会担心它们是如何生成的，然而，如果传送过来的是生成随机的 p 和 q 的 S 和 C 值，则可以自己来执行这个过程。使用单向哈希函数 SHA 是为防止别人逆推。

该法比 RSA 安全。在 RSA 中，素数是保密的，除非知道私钥，对别人是否生成一个伪造的

素数或者易于因子分解的某种特殊形式，你都一无所知。而在这一方法中，即使不知道私钥，也可以确证 p 和 q 是随机生成的。

用 DSA 的 ElGamal 加密

虽然美国政府青睐 DSA 是因为它只是一个数字签名算法，而不能用于加密，但是用 DSA 函数调用来执行 ElGamal 加密是可能的。

假设用一个函数调用实现 DSA 算法：

DSAsign(p,q,g,k,x,h,r,s)

函数接收参数 p,q,g,k,x 和 h，返回签名参数 r 和 s。

给报文 m 用公开密钥 y 执行 ElGamal 加密时，选择一个随机数 k，并调用：

DSAsign(p,p,g,k,0,0,r,s)

返回的 r 值是 ElGamal 算法中的 a，抛弃 s，再调用：

DSAsign(p,p,y,k,0,0,r,s)

把 r 更名为 u，抛弃 s，调用：

DSAsign(p,p,m,1,u,0,r,s)

抛弃 r，返回的 s 值是 ElGamal 算法中的 b，这样得到了密文 a 和 b。

解密时用秘密密钥 x，密文 a 和 b 调用：

DSAsign(p,p,a,x,0,0,r,s)

返回的 r 值为 $a^x \bmod p$，称之为 e，再调用：

DSAsign(p,p,1,e,b,0,r,s)

返回的 s 值为明文 m。

并非所有的 DSA 实现方案都可以这么做，有的要固定某些值，或者 p 和 q，或者某些参数的长度。但是，如果 DSA 实现方案相当一般，这仍不失为一种用数字签名库的加密方法。

用 DSA 的 RSA 加密

RSA 加密更加容易，用一个模数 n、报文 m 和公钥 e 调用：

DSAsign(n,n,m,e,0,0,r,s)

返回的 r 值为密文。

解密时调用：

DSAsign(n,n,m,d,0,0,r,s)

返回的 r 值为明文。

DSS 中难以觉察的信道

G. Simmons 发现了 DSS 中难以觉察的信道，这种信道使人们在他们的签名中嵌入秘密的报文，并只能由知道密钥的作者阅读。按照 Simmons 的说法，这是一个“值得注意的巧合”：DSS 中“使用 ElGamal 算法的难以觉察的信道的所有内在缺点都可克服”，并且 DSS“为目前已发现的难以觉察的通讯提供了最友好的设置”。由于难以觉察的信道使无所顾忌的 DSS 实现者在每个签名中泄露一点密钥，所以使用 DSS 的实现时要特别慎重。

13.6 ESIGN 算法

ESIGN 是由 T. Okamoto 等人设计的数字签名算法，按照设计者的观点，它对于类似的密钥和签名长度，至少同等安全并比 RSA 和 DSA 都快得多。

私钥为一对大素数 p 和 q，公钥为 $n=p^2q$。

H(x)为对报文 m 处理的哈希函数，另外还有一个安全性参数 k。

(1)A 选取一个小于 pq 的随机数 x。

(2)A 计算大于 $(H(m)-x^k \bmod n)/pq$ 的最小整数 w，并计算 $s=x+((w/kx^{k-1})\bmod p)pq$，把 s 送给 B。

(3)验证签名时，B 计算 $s^k \bmod n$，并计算大于 n 的比特数除以 3 的最小整数 a。如果H(m)小于等于 $s^k \bmod n$ 并且 $s^k \bmod n$ 小于 $H(m)+2^a$，则签名有效。

这个算法可以用预处理技术来加速：预处理可在任何时间执行并与待签名的报文无关。选取了 x 之后，A 把第(2)步分成二部分，第一部分可预计算。

(2a)A 计算：

$$u=x^k \bmod n$$
$$v=1/(kx^{k-1})\bmod p$$

(2b)A 计算：

$$w=\text{大于}(H(m)-u)/pq\text{ 的最小整数}$$
$$s=x+(wv \bmod p)pq$$

一般情况下，预计算可加速签名过程十倍，几乎所有的难工作都在预计算阶段完成了。另外还有用模算术运算技术来加速 ESIGN 算法，这个算法也可推广到椭圆曲线。

ESIGN 算法的安全性

ESIGN 算法最初提出时，k 值为 2，但很快被 E. Brickell 和 J. DeLaurentis 攻破，这二位攻击者后来也可攻破 k=3 的 ESIGN 算法。

设计者目前推荐 k 值采用 8,16,32,64,128,256,512 或 1024，也推荐 p 和 q 值至少各为 192 比特，使 n 至少为 576 比特，用这些参数，设计者推测 ESIGN 与 RSA 和 Rabin 同等安全。

13.7 McELIECE 算法

McEliece 基于代数编码理论设计了一种公钥密码体制，这个算法使用 Goppa 编码(一种纠错码)，他的思想是构造 Goppa 码并伪装成一般的线性码，Goppa 码有快速的解码算法，但一般线性码的解码是个难问题。

设 dH(x)为海明距离(Hamming distance)，n,k 和 t 值为系统参数。

私钥由二部分构成：G′为能纠正 t 个错误的 Goppa 码的 kxn 生成矩阵，P 为 n′xn 置换矩阵，S 为 k′x k 非奇异矩阵。

公钥为 k′ x k 的矩阵 G：$G=SG'P$。

明文报文为在GF(2)上 k 维矢量形式的 k 比特串。

加密报文时，随机选择一个 GF(2)上的 n 维矢量 z，其海明权重小于等于 t：

$$c=mG+z$$

解密时，先计算 $c'=cP^{-1}$，再用 Goppa 码的解码算法求解 $dH(m'G,c')<=t$ 的 m′，最后计算 $m=m'S^{-1}$。

McEliece 建议采用 n=1024，t=50 和 k=524。

虽然这个算法是最早的公钥算法之一，并且也没有对它成功的密码分析结果，但密码界从未广泛接受它。它比 RSA 要快 2～3 个数量级，但其中问题也不少，如公钥太大(2^{19}比特)，数据膨胀太大(密文是明文的二倍长)等。

1991 年，来自前苏联的二位学者 Korzhik 和 Turkin 攻破了这个系统，他们的攻击需要

$20n^3$ 次操作，对于 McEliece 建议的参数，可在约 60 小时内攻破系统。

13.8 OKAMOTO 92 算法

首次由 T. Okamoto 提出的这个签名算法由离散对数问题得出其安全保证。

有二个秘密素数 p 和 q，q 能除尽 p－1，q 值应至少 140 比特，p 至少 512 比特。另外还有二个随机数 g_1 和 g_2，都与 q 一样长。私钥为都小于 q 的一对随机数 s_1 和 s_2，公钥为 v：

$$v = g_1^{-s1} g_2^{-s2} \bmod p$$

签名报文 m 时，先选取二个都小于 q 的随机数 r_1 和 r_2，再计算：

$$e = H(g_1^{r1} g_2^{r2} \bmod p, m) \qquad \text{其中 } H(x) \text{ 为单向哈希函数}$$

$$y_1 = r_1 + es_1 \bmod q$$

$$y_2 = r_2 + es_2 \bmod q$$

签名为三重组(e, y_1, y_2)。

验证签名时，验证下式是否成立：

$$e = H(g_1^{y1} g_2^{y2} v^e \bmod p,\ m)$$

如果成立，则签名有效。

13.9 细胞自动机

这是 Guam 研究的、在公钥密码体制中采用细胞自动机的新思想，这个内容还未得到广泛研究，但已初步检查出可能有一些密码弱点，不过这还是很有前途的研究领域。

13.10 椭圆曲线密码体制

1985 年，N. Koblitz 和 V. S. Miller 各自提出把椭圆曲线用于公钥密码体制，他们没有设计出用椭圆曲线的新密码算法，但他们在有限域上用椭圆曲线实现了已有的公钥算法，如 Diffie-Hellman 算法。

椭圆曲线提供了一种构造“元素”及“组合规则”的方法，由它们可产生群，这些群有足够熟悉的特性来构造密码算法，但没有那些有助于密码分析的特性。例如，没有“平滑”的概念。

在 $GF(2^n)$ 有限域上的椭圆曲线特别有趣，n 在 130～200 范围内，特指域的算术处理过程，是易于建立也是相对地将较为简单的，这些系统有提供低成本的公钥密码体制的潜力。许多公钥算法，如 Diffie-Hellman，ElGamal 和 Schnorr 都可用有限域上的椭圆曲线实现。

13.11 其它公钥算法

过去数年中提出了又被攻破了许多公钥算法，如 Matsumoto-Imai 算法，Cade 算法的提出和被攻破。除了这些攻击方法之外，还有对有限域上分解多项式的攻击，任何有限域多项式组合算法提供的安全性都是值得怀疑的。

Yagisawa 算法把模 p 指数运算与模 p－1 算术运算组合，后来也被攻破，另外由 Tsujii 等人提出的公钥算法也是不安全的，Luccio-Mazzone 系统也是不安全的。

G. Simmone 提出 J-代数作为公钥算法的基础，在多项式分解的有效方法问世之后放弃了该思想，H. Niederreiter 提出了基于移位寄存器序列的公钥算法。

一些欧洲学者设计了用各种置换多项式代替指数的 RSA 的推广，这些变例与 RSA 同样安全，使用它们没有什么实际的安全益处。T. Okamoto 和 K. Ohta 把一些数字签名算法作了

比较。

设计出全新不同的公钥密码算法似乎不太乐观，1988 年，Diffie 认为大多数公钥算法都基于以下三个难问题之一：

1. 背包问题

给出一个由各不相同的数构成的集合，求一个和为 N 的子集。

2. 离散对数

如果 p 为素数，g 和 M 为整数，求解满足 $g^x = M \pmod p$ 的 x。

3. 因子分解

如果 N 为二个素数的乘积，

(a)因子分解 N

(b)给定二个整数 M 和 C，求满足 $M^d = C \pmod N$ 的 d。

(c)给定二个整数 e 和 C，求满足 $M^e = C \pmod N$ 的 M。

(d)给定一个整数 x，确定是否存在满足 $x = y^2 \pmod N$ 的整数 y。

Diffie 认为，离散对数问题由 J. Gill 提出，因子分解问题由 Knuth 提出，背包问题由他本人提出。

公钥密码术的数学基础是很狭窄的，无论是因子分解还是计算离散对数问题的突破都会使整类公钥算法不安全。

并非所有基于这些问题的公钥算法都是安全的，任何公钥算法的强度更依赖于相应问题的计算复杂性，一个计算困难的算法未必是个强算法，Shamir 列出了其中的三个原因：

1. 复杂性理论往往处理问题的单个独立事例，密码分析者往往拥有大量统计相关的问题解答——许多密文以同一密钥加密。

2. 问题的计算复杂性典型地由其最坏或平均情况来度量，而在密码术中，问题必须在所有情况下都难解。

3. 任意计算机难解的问题未必能用于密码系统，必须能够在问题中插入陷门信息，有且仅有这些信息才能简捷地实现。

13.12 哪个公钥算法最优？

这个问题不仅与不同的算法完成不同的功能有关，而且还涉及它们的专利等问题。对于加密和数字签名，RSA 是最易于实现的。ElGamal 算法更适合于加密，DSA 对数字签名是极好的，并且 DSA 无专利税，可以随意获取。Diffie-Hellman 是最容易的密钥交换算法，各种身份认证算法也各有其用途。不过在采用其它算法之前，最好查阅一下资料以确保它未被攻破。

第 14 章 单向哈希函数

14.1 背 景

单向哈希函数 H(M)对一个任意长度的报文 M 施加操作，返回一个固定长度的哈希值 h：

h＝H(M)　　　　h 的长度为 m

许多函数都接受任意长度的输入而返回固定长度的输出，但单向哈希函数具有下述不同的特征：

- 给定 M，易于计算 h
- 给定 h，计算 M 是很困难的
- 给定 M，找到满足 H(M)＝H(M′)的另一个报文 M′是很困难的

“困难”联系着特定的安全要求，但大多数现实应用中都把“困难”定义为需要 264 次操作，偶尔会更多。

对单向哈希函数有二种基本的攻击方法，第一种是直截了当的方法：给定报文的哈希值 H(M)，敌手建立另一个文档 M′，使得 H(M)＝H(M′)。如果攻击成功，就会破坏使用单向哈希函数的每个协议的安全性。单向哈希函数的要点是提供 M 的唯一“指纹”，如果 A 用数字签名算法对报文 M 的签字是签在它的 H(M)上，B 可以产生满足 H(M)＝H(M′)的 M′，则 B 就可以声称 A 签署了 M′。

另一种攻击方法(生日攻击)比较微妙，因为找到二个满足 H(M)＝H(M′)的随机报文 M 和 M′ 必定是很困难的，相比之下，生日攻击比第一种方法来得容易。

生日悖论是个标准的统计问题：一个房间中要有多少人才能使有人与你生日相同的概率大于平均值？答案为 183。那么，有多少人时才能使其中至少有二人生日相同的概率大于平均值？答案是只需少数人：23，房间中只要有 23 人即可，但是其中有 500 多“对”是不同的。

按特定生日找到某人类似于第一种攻击，找到任一相同生日的二人类似于第二种攻击，第二种攻击一般称为生日攻击(birthday attack)。

假设一个单向哈希函数遵循上述所有特性，并且最好的攻击方法是蛮力攻击，它产生 m 比特的输出。找到一个哈希散列到给定值的报文需要哈希散列 2^m 个随机报文，找到二个 100 万哈希值相同的报文只需 $2^{m/2}$ 个随机报文。一台每秒哈希散列 100 万报文的机器需要 600,000 年才能找到与给定 64 比特值匹配的第二个报文，同一台机器找到一对哈希值相同的报文只需约 1 个小时。

由 Yuval 首次提出了如何使用生日攻击来骗取对方的方案。

(1) A 准备二份合同，一份是有利于 B 显然他会同意的，另一份将取走他的所有财产而会被拒绝的。

(2) A 在每个文档中作了几处小修改并计算各个的哈希值(这些修改可以是用空格-回退符-空格代替空格，在回车之前加一或二个空格等等，在 32 行的每行中或作或不作某个修改，A 可方便地生成 2^{32} 个不同的文档)。

(3) A 比较每二个文档的哈希值，寻找一个匹配对（如果哈希函数只输出一个 64 比特的值，她往往用 2^{32} 个值可以找到一个匹配对），她把哈希值相同的二个报文重新构造。

(4) A 让 B 在有利于他的那份合同上签字，B 用某个哈希值签了字。

(5) 以后某时，A 用 B 没有签字的合同替换了他签了字的合同，她可以让仲裁人确信 B 在这一份合同上也签了字。

单向哈希函数的长度

64 比特的哈希函数对于防生日攻击来说是太小了，大多数的实际的单向哈希函数生成 128 比特的哈希值，这样试图生日攻击的人就得哈希散列 2^{64} 个随机文档才能找到二个哈希值相同的文档。NIST 的安全哈希标准(Secure Hash Standard, SHS) 中采用 160 比特的哈希值，这使生日攻击更加困难，需要 2^{80} 次随机哈希散列。

生成比哈希函数的结果更长的哈希值，可有下面的方法：

(1) 用某单向哈希函数生成报文的哈希值。

(2) 把哈希结果附加在报文上。

(3) 生成报文和哈希结果连接的哈希值。

(4) 建立一个大的哈希值，由第(1)步和第(3)步生成的哈希值连接而成。

(5) 任意次重复第(1)～(3)步。

单向哈希函数概况

有许许多多设计单向哈希函数的文章，Preneel 的博士论文可能是单向函数最全面的论述。大多数单向哈希函数都建立在给定二个长为 n 的输入、输出一个长为 n 的值的单向函数之上。函数的输入一般为文本块与文本前一块的哈希值（见图 14.1)，因此块 Mi 的哈希散列为：

$$h_i = f(M_i, h_{i-1})$$

最后一块的哈希值成为整个报文的哈希值。

这样，无论输入的大小如何，单向哈希函数总是形成一个固定长度的输出。

典型的哈希散列了的信息为全报文长度的某种二进制表示，此项技术克服了不同长度的报文可能哈希散列到同一值而造成不安全的隐患。

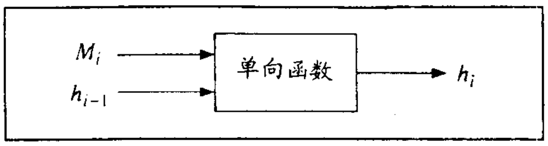


图 14.1　单向哈希函数

许多研究者都证明了如果单向哈希函数对某个块是安全的，则把任意长度的块串作哈希散列的方法也是安全的。

14.2　SNEFRU 算法

Snefru 算法是由 R. Merkle 设计的单向哈希函数，它把任意长度的报文哈希散列到 128 或 256 比特之值。

首先把报文以(512－m)的大小分块(m 为哈希值的长度)，如果输出为 128 比特的哈希值，则每块的长度为 384 比特，如果输出为 256 比特的哈希值，则每块的长度为 256 比特。

算法的核心是函数 H，它把 512 比特的数据哈希散列为 m 比特。H 函数的设计思想是：由输入计算哈希值是容易的，但对给定哈希值去计算其相应的输入则是不可行的。

H 函数输出的前 m 比特为本块的哈希值，其余不用。下一块附在前一块哈希值的后面，并再次哈希散列(初始块后面附 0)。处理了最后一块之后(如果报文长度不是块长的整数倍，则

最后一块为短块，也以 0 填满。前 m 比特数据附在该以 0 填补的报文之后，并执行最后一次哈希散列。

函数 H 基于另一个函数 E，函数 E 是可逆块密码函数，它作 512 比特的块操作。H 为 E 输出的最后 m 比特与 E 输入的前 m 比特的异或。

Snefru 算法的安全性在于函数 E，函数 E 多遍随机化数据，每遍由 64 轮随机化组成，每轮中采用数据的不同字节作为 S 盒的输入，S 盒的输出字与报文的二个相邻字异或，S 盒的构造方法类似于 Khafre 算法。原来的 Snefru 算法中只有二遍，后来增加了一些循环处理。

Snefru 算法密码分析

Biham 和 Shamir 用差分密码分析技术说明了二遍的 Snefru 算法(128 比特哈希值)是不安全的，他们出示了哈希值相同的下列报文：

3FE15E26	23B7C030	C7089999	90EFC48F	A04D87EE	16493392
00046085	00003415	00000000	00000000	00000000	00000000
3FE15E26	23B7C030	C7089999	90EFC48F	A04D87EE	D74AF7AE
096C7885	C19EF029	00000000	00000000	00000000	00000000

共同的哈希值为：C8FF5E2C　8F9CF7C7　F08DDAA7　E4F9B44E

下列四个报文的哈希值也相同：

00000000	00000000	00000000	00000000	00000000	00000000
00000000	00000000	00000000	00000000	00000000	00000000
00000000	F1301600	13DFC53E	4CC3B093	37461661	CCD8B940
24D9D35F	71471FDE	00000000	00000000	00000000	00000000
00000000	1D197F00	2ABD3F6F	CF33F3D1	8674966A	816E5D51
ACD9A905	53C1D180	00000000	00000000	00000000	00000000
00000000	E98C8300	1E777A47	B5271F34	A04974BB	44CC8B62
BE4B0EFC	18131756	00000000	00000000	00000000	00000000

共同的哈希值为：2E88E244　E9D4A208　B2DD2FBB　72D0EEE6

在 PC 机上，他们在 3 分钟内找到了哈希值相同的一对报文，并在约 1 个小时内找到哈希散列到给定值的一个报文。

对于 4 遍以下的 128 比特的 Snefru 算法，他们的攻击方法比蛮力攻击好。对 Snefru 的生日攻击需要 2^{64} 次操作，而差分分析法对于 3 遍的 Snefru 算法可在 $2^{28.5}$ 次操作内找到哈希值相同的一对报文，对于 4 遍的 Snefru 算法只需 $2^{44.5}$ 次操作。找到一个哈希散列到给定值的报文，蛮力攻击需要 2^{128} 次操作，而差分分析法对于 3 遍的 Snefru 算法只需 2^{56} 次操作，对于 4 遍的 Snefru 算法需 2^{88} 次操作。

对于更长哈希值长度的 Snefru 算法，结果也比蛮力攻击好，虽然 Biham 和 Shamir 没有分析 256 比特的哈希值，但他们把分析推广到 224 比特的哈希值。相对于生日攻击需要 2^{112} 次操作，他们对 2 遍的 Snefru 算法可在 $2^{12.5}$ 次操作内找到一对哈希值相同的报文，对于 3 遍的 Snefru 算法只需 2^{33} 次操作，对于 4 遍的 Snefru 算法只需 2^{81} 次操作。

目前,Merkle 推荐采用至少 8 遍的 Snefru 算法,然而,这样的算法会比 MD5 和 SHA 慢得多。

14.3 N-哈希算法

N-哈希算法是由 Nippon 电话电报公司的研究者设计的算法,与 FEAL 算法同出一人之手。它于 1990 年首次呈递给密码界。

N-哈希算法使用 128 比特的报文块,一个类似于 FEAL 的复杂随机化函数,并产生一个 128 比特的哈希值。

每个 128 比特块的哈希值是该块与前一块哈希值的函数:

$h_0 = I$　　I 为随机的初值

$h_i = g(M_i, h_{i-1})$ XOR M_i XOR H_{i-1}

整个报文的哈希值为最后一个报文块的哈希值,随机初值 I 为用户决定的任意值(甚至全 0)。

函数 g 是个复杂函数(见图 14.2,14.3 和 14.4)。图 14.2 是算法概况,开始时前一报文块的 128 比特哈希值 h_{i-1}左半部分与右半部分(各为 64 比特)交换,再与二进制的 1010…1010 (128 比特)异或,再与当前报文块 M_i 异或,异或结果经过 N 级(图中 N=8)处理,每级处理的另一个输入是前一哈希值与 8 个二进制常量之一的异或结果。

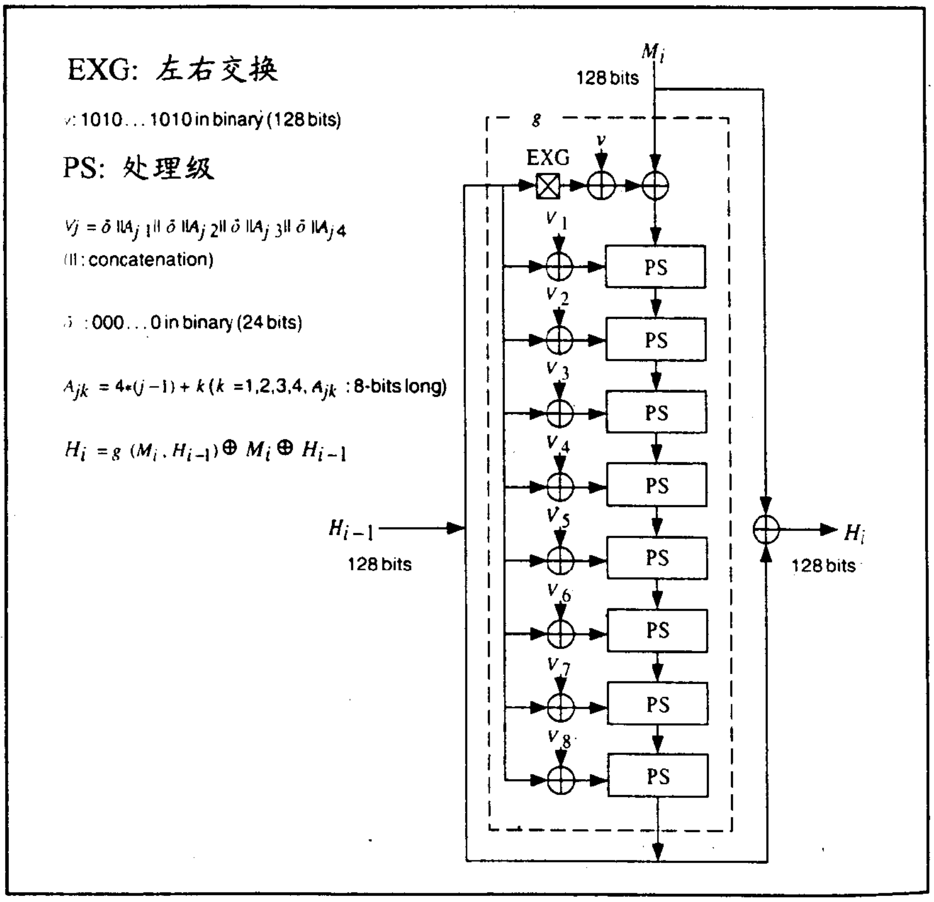


图 14.2 N-哈希概况

图 14.3 为一级处理。报文分成 4 个 32 比特的值,前一哈希值也分成 4 个 32 比特的值。函数 f 如图 14.4 所示,函数 S_0 和 S_1 与 FEAL 算法中相同:

$S_0(a,b) = ((a+b) \bmod 256)$循环左移 2 位

$S_1(a,b) = ((a+b+1) \bmod 256)$循环左移 2 位

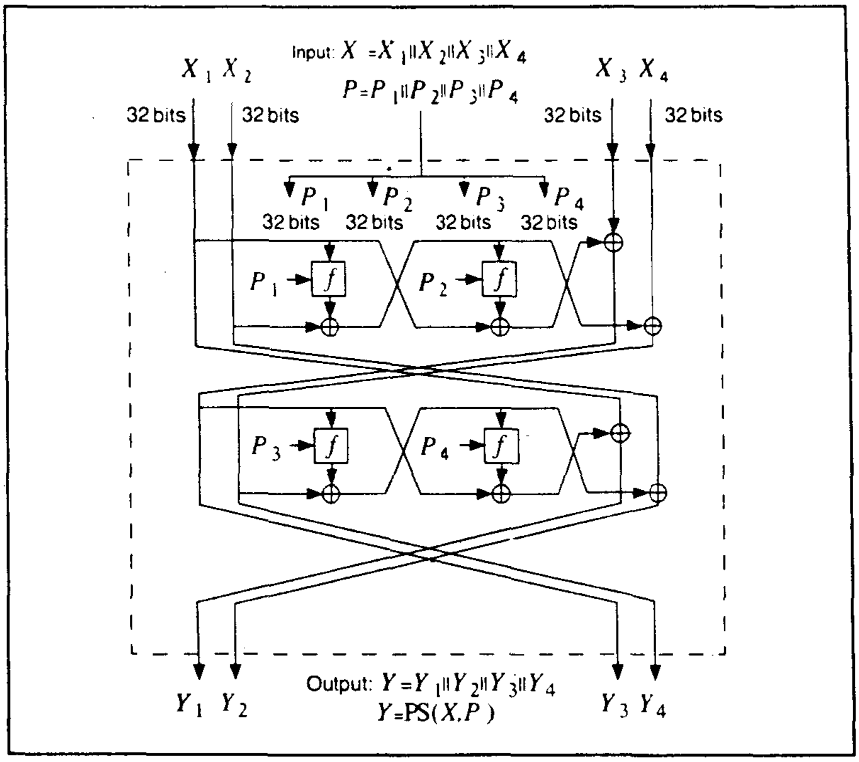


图 14.3 N-哈希的一级处理

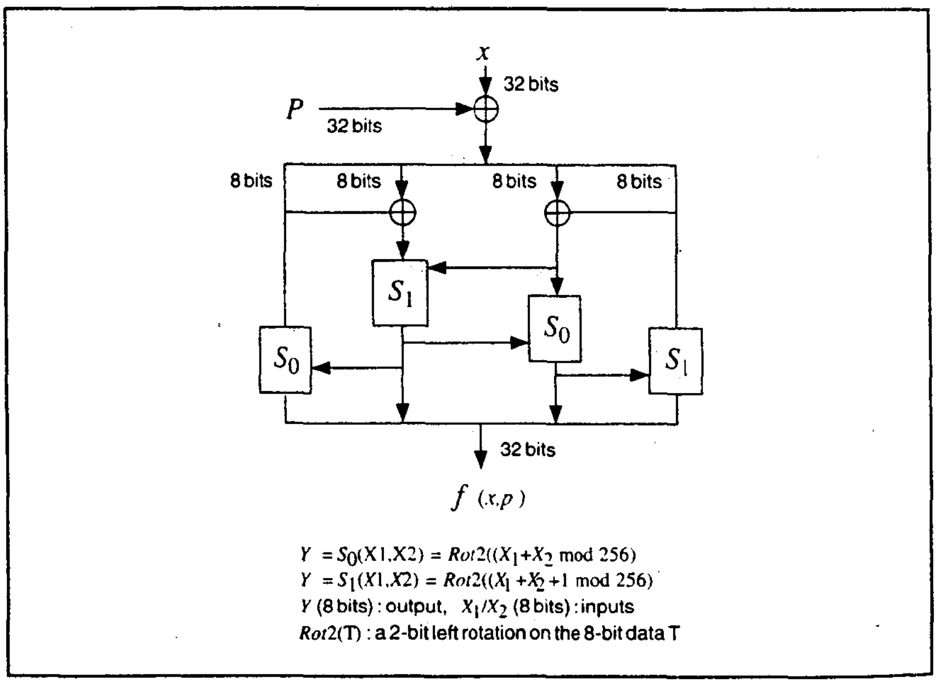


图 14.4 函数 f 的结构

一级处理的输出成为下一级处理的输入，最后一级处理之后，输出 S 与 M_i 和 h_{i-1} 异或，接着准备对下一块哈希散列处理。

N-哈希算法密码分析

Biham 和 Shamir 用差分分析法攻破了 6 轮的 N-哈希算法，他们特殊的攻击方法针对任何 3 的倍数的 N。对于小于 15 的 N，他们的方法比生日攻击更有效。

对于 6 轮的 N-哈希算法，Biham 和 Shamir 发现下列报文的哈希值相同：

CAECE595 127ABF3C 1ADE09C8 1F9AD8C2

4A8C6595 921A3F3C 1ADE09C8 1F9AD8C2

共同的哈希值为:12B931A6 399886B7 640B9289 36C2EF1D

5878BE49 F2962D67 30661E17 0C38F35E

D8183E49 72F6AD67 30661E17 0C38F35E

共同的哈希值为:29B0FE97 3D179E0E 5B147598 137D28CF

同一方法对 12 轮的 N-哈希算法可在 2^{56}次操作内找到哈希值相同的密文对,相对于蛮力攻击需要 2^{64}次操作。15 轮的 N-哈希算法对于差分分析是安全的:需要 2^{72}次操作。

14.4 MD4 算法

MD4 是由 R. Rivest 设计的单向哈希函数,MD 表示"Message Digest(报文提要)",算法产生输入报文的 128 比特的哈希值,或称为报文提要。

Rivest 概括了算法的设计目标:

· 安全性。找到二个哈希值相同的报文是计算上不可行的,没有比蛮力攻击更有效的攻击方法。

· 直接安全性。MD4 的安全性不建筑在如因子分解的难度等任何假设上。

· 速度。MD4 适合于高速的软件实现,它基于对 32 比特操作数的一些简单的位操作。

· 简单性。MD4 尽可能简单,没有大的数据结构或复杂的程序。

· 偏爱 Little-Endian 体系结构。MD4 为微处理器体系结构(尤其是 Intel 微处理器)作了优化,更大和更快的计算机作必需的转换。

算法首次提出后,B. Boer 和 A. Bosseleass 成功地密码破译了算法的三轮中的二轮,R. merkle 成功地攻击了前二轮,E. Biham 对 2/3 的 MD4 讨论了可能的差分分析,虽然这些攻击未能扩展到整个算法,但 Rivest 巩固了算法,产生了 MD5。

14.5 MD5 算法

MD5 是 MD4 的改进版,虽然比 MD4 更复杂,但设计是类似的,并也产生 128 比特的哈希值。

算法描述

在初始化处理之后,MD5 处理分成 16 个 32 比特子块的 512 比特输入文本块,算法输出为 4 个 32 比特的块,连接成一个 128 比特的哈希值。

首先填充报文,使得报文长度为 512 的倍数 64 比特,填充是在报文末尾加一个 1 和若干个 0,再在结果后面附上填充之前的报文长度的 64 比特表示。这二步使报文长度为 512 比特的整倍数(为算法后续处理),从而保证不同的报文在填充之后不一样。

初始化了如下 4 个 32 比特的变量:

A=01234567

B=89ABCDEF

C=FEDCBA98

D=76543210

这些变量称为链接变量(chaining variable)。

于是开始算法的主循环，循环执行次数为报文中 512 比特的块数次。

4 个变量复制到不同的变量：A 到 AA，B 到 BB，C 到 CC，D 到 DD。

主循环有 4 轮（MD4 只有 3 轮），每轮有 16 个操作，每个操作执行 A，B，C 和 D 中 3 个变量的非线性函数，再把结果加到第 4 个变量：文本的一个子块及一个常量，再将结果循环右移若干位（由某变量指定）并把结果加到 A，B，C 和 D 中之一，最后把结果替换 A，B，C 和 D 中之一。

4 个非线性函数如下，每一轮中采用一个：

F(X,Y,Z)=XY or(not X)Z

G(X,Y,Z)=XZ or Y(not Z)

H(X,Y,Z)=X XOR Y XOR Z

I(X,Y,Z)=Y XOR(X or(not Z))

这些函数的设计宗旨是如果 X，Y 和 Z 中相应的各位是独立的和公正的，则结果的每一位也是独立的和公正的。函数 F 为位方式条件：IF X then Y else Z，函数 H 为位方式奇偶性操作符。

如果 M_j 代表报文的第 j 个子块（j=0…15），并且<<s 表示左移 s 位，4 个操作为：

FF(a,b,c,d,M_j,s,t_j)表示 a=b+((F(b,c,d)+M_j+t_i)<<s)

GG(a,b,c,d,M_j,s,t_j)表示 a=b+((G(b,c,d)+M_j+t_i)<<s)

HH(a,b,c,d,M_j,s,t_j)表示 a=b+((H(b,c,d)+M_j+t_i)<<s)

II(a,b,c,d,M_j,s,t_j)表示 a=b+((I(b,c,d)+M_j+t_i)<<s)

4 轮（64 步）操作如下：

第 1 轮：

FF(a,b,c,d,M[0],7,0xd76aa478)

FF(d,a,b,c,M[1],12,0xe8c7b756)

FF(c,d,a,b,M[2],17,0x242070db)

FF(b,c,d,a,M[3],22,0xc1bdceee)

FF(a,b,c,d,M[4],7,0xf57c0faf)

FF(d,a,b,c,M[5],12,0x4787c62a)

FF(c,d,a,b,M[6],17,0xa8304613)

FF(b,c,d,a,M[7],22,0xfd469501)

FF(a,b,c,d,M[8],7,0x698098d8)

FF(d,a,b,c,M[9],12,0x8b44f7af)

FF(c,d,a,b,M[10],17,0xffff5bb1)

FF(b,c,d,a,M[11],22,0x895cd7be)

FF(a,b,c,d,M[12],7,0x6b901122)

FF(d,a,b,c,M[13],12,0xfd987193)

FF(c,d,a,b,M[14],17,0xa679438e)

FF(b,c,d,a,M[15],22,0x49b40821)

第 2 轮：

GG(a,b,c,d,M[1],5,0xf61e2562)

GG(d,a,b,c,M[6],9,0xc040b340)

GG(c,d,a,b,M[11],14,0x265e5a51)
GG(b,c,d,a,M[0],20,0xe9b6c7aa)
GG(a,b,c,d,M[5],5,0xd62f105d)
GG(d,a,b,c,M[10],9,0x02441453)
GG(c,d,a,b,M[15],14,0xd8a1e681)
GG(b,c,d,a,M[4],20,0xe7d3fbc8)
GG(a,b,c,d,M[9],5,0x21e1cde6)
GG(d,a,b,c,M[14],9,0xc33707d6)
GG(c,d,a,b,M[3],14,0xf4d50d87)
GG(b,c,d,a,M[8],20,0x455a14ed)
GG(a,b,c,d,M[13],5,0xa9e3e905)
GG(d,a,b,c,M[2],9,0xfcefa3f8)
GG(c,d,a,b,M[7],14,0x676f02d9)
GG(b,c,d,a,M[12],20,0x8d2a4c8a)

第 3 轮：

HH(a,b,c,d,M[5],4,0xfffa3942)
HH(d,a,b,c,M[8],11,0x8771f681)
HH(c,d,a,b,M[11],16,0x6d9d6122)
HH(b,c,d,a,M[14],23,0xfde5380c)
HH(a,b,c,d,M[1],4,0xa4beea44)
HH(d,a,b,c,M[4],11,0x4bdecfa9)
HH(c,d,a,b,M[7],16,0xf6bb4b60)
HH(b,c,d,a,M[10],23,0xbebfbc70)
HH(a,b,c,d,M[13],4,0x289b7ec6)
HH(d,a,b,c,M[0],11,0xeaa127fa)
HH(c,d,a,b,M[3],16,0xd4ef3085)
HH(b,c,d,a,M[6],23,0x04881d05)
HH(a,b,c,d,M[9],4,0xd9d4d039)
HH(d,a,b,c,M[12],11,0xe6db99e5)
HH(c,d,a,b,M[15],16,0x1fa27cf8)
HH(b,c,d,a,M[2],23,0xc4ac5665)

第 4 轮：

II(a,b,c,d,M[0],6,0xf4292244)
II(d,a,b,c,M[7],10,0x411aff97)
II(c,d,a,b,M[14],15,0xab9423a7)
II(b,c,d,a,M[5],21,0xfc93a039)
II(a,b,c,d,M[12],6,0x655b59c3)
II(d,a,b,c,M[3],10,0x8f0ccc92)
II(c,d,a,b,M[10],15,0xffeff47d)
II(b,c,d,a,M[1],21,0x85845dd1)

II(a,b,c,d,M[8], 6,0x6fa87e4f)
II(d,a,b,c,M[15],10,0xfe2ce6e0)
II(c,d,a,b,M[6],15,0xa3014314)
II(b,c,d,a,M[13],21,0x4e0811a1)
II(a,b,c,d,M[4], 6,0xf7537e82)
II(d,a,b,c,M[11],10,0xbd3af235)
II(c,d,a,b,M[2],15,0x2ad7d2bb)
II(b,c,d,a,M[9],21,0xeb86d391)

其中常量 t_i 选择如下：

第 i 步中，t_j 为 4294967296xabs(sin(i))的整数部分，其中 i 为弧度（注意 4294967296 为 2^{32}）。

所有这些操作完成之后，A，B，C 和 D 各自加到 AA，BB，CC 和 DD 中，算法继续处理数据的下一块。最后的输出为 A，B，C 和 D 的连接。

MD5 算法的安全性

R. Rivest 概括了 MD5 对于 MD4 的改进：

1. 增加了第 4 轮。
2. 每一步都用唯一的加法常量。
3. 第 2 轮中的函数 g 由(XY or XZ or YZ)改为(XZ or Y(not Z))，使 g 更不对称。
4. 每一步中加入前一步的结果，从而促使更快的"雪崩效应"。
5. 修改了第 2 轮和第 3 轮中处理输入字的次序，减少这些模式相象程度。
6. 优化了各轮中的移位位数，以形成更快的"雪崩效应"，各轮中移位互不相同。

T. Berson 曾试图对 MD5 的单轮作差分分析，但他的攻击对 4 轮有效还相距甚远。B. Boer 和 A. Bosselaers 在 MD5 中用压缩函数产生混乱，这并不有助于攻击实际应用中的 MD5：它的确意味着 MD5 的基本设计原则之一，违反了设计一个抗混乱的压缩函数。

14.6 MD2 算法

MD2 是 R. Rivest 设计的另一个单向哈希函数，它和 MD5 一起用于 PEM 协议，它对任意的输入产生一个 128 比特的哈希值，其结构上类似于 MD4 和 MD5，速度更慢，安全性也较差。

14.7 安全哈希算法

NIST 和 NSA 一起设计了安全哈希算法(Secure Hash Algorithm，SHA)与 DSA 一起使用。

一种安全哈希标准(SHS)正被提为联邦信息处理标准(FIPS)，这个标准指定安全哈希算法(SHA)与数字签名标准配套使用，另外，对于不需要数字签名的应用场合，无论何时联邦应用需要安全哈希算法时都采用 SHA。

这个标准指定 SHA 保证 DSA 的安全性，当输入长度小于 2^{64} 比特的报文时，SHA 产生称为"报文提要"的 160 比特输出，报文提要再输入 DSA 计算报文的签名。签名报文提要比签名报文更有效，因为前者往往比后者小得多。当接收到的报文用作 SHA 的输入时，签名的检验者应得到同样的报文提要。SHA 称为安全的是因为其设计原则是找到一个相应于给定报文提要的报文，或者找到二个产生同样报文提要的不同报文是计算上不可行的。传输中报文的任何

修改很可能导致不同的报文提要，并且签名会检验失败。SHA 基于类似于 MD4 的设计原则，并严格由这个算法建模。

SHA 产生 160 比特的哈希值，比本章中的其它算法都长。

SHA 算法描述

首先将报文填充成长度为 512 比特的倍数，填充方法与 MD5 完全相同：先附加一个 1，再若干个 0，使长度为 512 的倍数－64，最后是填充之前报文长度的 64 比特表示。

初始化如下 5 个 32 比特的变量：

A＝67 45 23 01

B＝EF CD AB 89

C＝98 BA DC FE

D＝10 32 54 76

E＝C3 D2 E1 F0

开始执行算法的主循环，每次处理 512 比特的报文，并且重复次数为报文中 512 比特的块数次。

把 5 个变量复制到不同的变量：A 到 AA，B 到 BB，C 到 CC，D 到 DD，E 到 EE。

主循环有 4 轮，每轮有 20 个操作，每个操作执行 A，B，C 和 D 中 3 个变量的非线性函数，再做类似于 MD5 中的移位和相加。

SHA 的非线性函数为：

$f_t(X,Y,Z)=XY$ or (not X)Z　　第 1 个 20 个操作

$f_t(X,Y,Z)=X$ XOR Y XOR Z　　第 2 个 20 个操作

$f_t(X,Y,Z)=XY$ or XZ or YZ　　第 3 个 20 个操作

$f_t(X,Y,Z)=X$ XOR Y XOR Z　　第 4 个 20 个操作

算法中也使用了 4 个常量：

K_t＝5A827999　　第 1 个 20 个操作

K_t＝6ED9EBA1　　第 2 个 20 个操作

K_t＝8F1BBCDC　　第 3 个 20 个操作

K_t＝CA62C1D1　　第 4 个 20 个操作

报文用下面算法从 16 个 32 比特字（$M_0 \sim M_{15}$）变换到 80 个 32 比特字（$W_0 \sim W_{79}$）：

$W_t=M_t$　　$t=0,\cdots,15$

$W_t=W_{t-3}$ XOR W_{t-8} XOR W_{t-14} XOR W_{t-16}　　$t=16,\cdots,79$

如果 t 为操作次数（1－80），M_j 表示第 j 个报文子块（0－15），<<s 表示左移 s 位，则 80 个操作为：

$TEMP=(A<<s)+f_t(B,C,D)+E+W_t+K_t$

E＝D

D＝C

C＝B

B＝A

A＝TEMP

所有这些操作结束之后，A，B，C，D 和 E 分别加到 AA，BB，CC，DD 和 EE 中，算法继续处理数据的下一块。最后的输出为 A，B，C，D 和 E 的连接。

SHA 的安全性

SHA 非常类似于 MD4，主要改变为：增加了扩展转换，增加了把前一轮的输出加到下一轮，以加快"雪崩效应"，提高了整个转换以适合 DSS 的块大小。SHA 和 MD5 一样是 MD4 的变例而不是重新设计。

R. Rivest 公开了 MD5 的设计原则，但 SHA 的设计者没有公开。以下是 Rivest 的 MD5 对 MD4 的改进以及与 SHA 的比较：

1. "增加了第 4 轮"，SHA 也如此。然而，SHA 中第 4 轮与第 2 轮使用同样的 f 函数。

2. "每一步都用唯一的加法常量"，SHA 保持 MD4 的方法，每 20 轮中采用同一常量。

3. "第 2 轮中的函数 g 由(XY or XZ or YZ)改为(XZ or Y(not Z))，使 g 更不对称"，SHA 采用 MD4 的方法：(XY or XZ or YZ)。

4. "每一步中加入前一步的结果，从而促使更快的'雪崩效应'"，SHA 中也做了如此更改，SHA 中的区别是增加了第 5 个变量，并且不是 fi 中使用的 B，C 或 D，这个微妙的更改使得 Boer-Bosselaers 对 MD5 的攻击方法对 SHA 无效。

5. "修改了第 2 轮和第 3 轮中处理输入字的次序，减少这些模式相象程度"，SHA 是完全不同的，它采用循环纠错码。

6. "优化了各轮中的移位位数，以形成更快的'雪崩效应'，各轮中移位互不相同"，SHA 在每轮中移位位数恒定，与 MD4 一样，是与字大小互素。

因此，可以说

SHA ＝ MD4 ＋"扩展"变换 ＋ 额外轮 ＋ 更好的"雪崩效应"

MD5 ＝ MD4 ＋ 改进的位哈希散列 ＋ 额外轮 ＋ 更好的"雪崩效应"

目前还没有对 SHA 的已知密码攻击，并且由于它产生 160 位的哈希值，它比其它算法更能抗蛮力攻击(包括生日攻击)。

14.8 RIPE-MD 算法

RIPE-MD 算法是由欧共体 RACE 项目设计的，它是 MD4 的变例，为抗已知的密码攻击而设计。修改了循环移位和报文字的次序，另外，只是常量不同的算法二个例程并行运行，处理完每一块之后，二个例程的输出加到链接变量。

14.9 HAVAL 算法

HAVAL 算法是 Yuliang Zhang 等人设计的可变长的单向哈希函数，它是 MD5 的改进形式，HAVAL 算法处理 1024 比特的报文块，为 MD5 的二倍。它有 8 个 32 比特的链接变量，也是 MD5 的二倍。它的轮数从 3 到 5 可变，每轮有 16 步，并产生 128，160，92，224 和 256 比特的哈希值。

HAVAL 算法用高度非线性的 7 变量函数代替 MD5 中的简单非线性函数，每轮用一个函数，但每步中不同的置换用于输入。报文次序也不同，并且每步(除第 1 轮外) 采用不同的加法常量，同样也有二种循环移位。

算法核心为：

TEMP＝(f(j，A，B，C，D，E，F，G)<<7)＋(H<<11)＋M[i][r(j)]＋K(j)

H＝G

G＝F

F=E

E=D

D=C

C=B

B=A

A=TEMP

HAVAL 的速度比 MD5 快：3 轮快 60%，4 轮快 15%，5 轮同样快。

可变的轮数和可变的输出长度是指算法有 15 种不同的版本。由于 H 作了循环移位，D. Boer和 Bosselaers 对 MD5 的攻击方法不能用于 HAVAL 算法。

14.10 其它单向哈希函数

滑铁卢大学的一组研究人员提出了基于 GF(2^{593}) 上迭代指数的单向哈希函数，他们把报文以 563 比特分块，各块从第 1 块开始依次取指数，每个指数是前一块的计算结果，第一个指数由初始化矢量给定。

I. Damgard 基于背包问题设计的单向哈希函数可在约 2^{32}次操作内攻破。

S. Wolfram 的细胞自动机已作为单向哈希函数的基础，但 I. Damgard 的早期实现是不安全的，基于细胞自动机的单向哈希函数有 Cellhash 及其改进版 Subhash。

C. P. Schnorr 在 1991 年夏提出一种基于离散傅立叶变换的单向哈希函数，称为 FFT-Hash，数月之后二个独立的研究团体攻破了它，Schnorr 提出了修正版，称为 FFT-Hash II，数月之后又被攻破，Schnorr 又提出了进一步的改进，但是要比其它算法慢得多。

14.11 使用对称块算法

对称块密码算法是可能用作单向哈希函数的，其思想是如果知道块算法是安全的，则单向哈希函数也是安全的。

最直观的方法是采用 CBC 方式的算法，把 IC 设为 0 并把最后一个密文块处理成哈希值，FIPS 113 中描述了用 DES 的这种方法。

然而，这个方法有二个问题：DES 和大多数块算法生成 64 比特的块，这对于可用的哈希值来说往往是太小了。另外，这个方法必须为算法指定一个加密密钥，并保密好密钥，否则，攻击者可以反过来运行算法：用已知的密钥解密哈希值。

聪明一点就是用报文块作为密钥，前一哈希值作为输入，当前哈希值作为输出。

哈希函数的细节十分复杂，大量的异或运算和内容添加意味着块的大小与密钥长度相同，哈希值的长度与块的大小一样。由于大多数块算法都是 64 比特的，围绕着二倍块大小的哈希散列出现了许多设计方案。

Davies-Meyer 算法

这一算法分别由 Davies 和 Meyer 提出，可用于任何块密码算法，哈希值的长度为本书中大多数块算法中的块大小——64bit。

在每轮中，用下列公式把报文的一块作哈希散列：

$H_0=I$　　　　I 为随机初始值

$H_i=E_{M_i}(H_{i-1})\ \mathrm{XOR}\ H_{i-1}$

图 14.5 是这一方法的说明，报文块 M_i 用作加密密钥，前一报文块用该密钥加密并与自身异或

(这似乎增加了一个多余的异或操作来抵抗中间会合攻击),该结果成为下一轮的输入。最后报文的长度成为最后的 M_i。

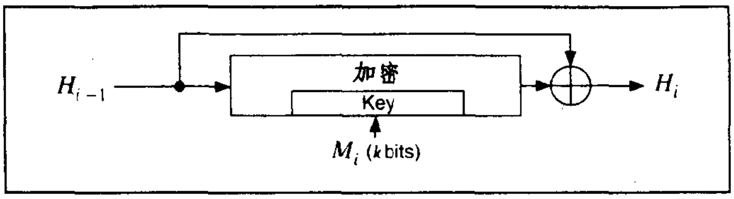


图 14.5 Davies-Meyer 算法结构

该单向哈希函数与基本加密算法一样安全。它只产生 64 比特的哈希值,所以它不适用于对生日攻击犯忌的应用场合。

采用 DES 时,首先将 M 分成各为 56 比特(DES 密钥的长度)的块,第 1 个报文块 M_1 成为 DES 密钥,DES 算法的明文输入为随机初值 H_0,DES 算法的密文输出为 H_1,H_1 与 H_0 异或成为第 2 块哈希散列的输入。下一个 56 比特的报文块 M_2 成为新的 DES 密钥,H_1 为算法的明文输入,H_2 为密文输出,H_2 与 H_1 的异或结果成为第 3 块哈希散列的输入。如此重复。

由于 DES 算法为每块都采用新的密钥,这个算法比前述所有单向哈希函数都慢,每次都要生成所有子钥,并且更坏的是,这个算法只生成 64 比特的哈希值。

LOKI 单块算法

这一算法由 LOKI 算法的创始人设计,它生成等于块长的哈希值。

$H_0 = I_H$ I_H 为随机初始值

$H_i = E_{H_{i-1} XOR M_i}(H_{i-1}) XOR H_{i-1} XOR M_i$

图 14. 6 说明了这个算法,由于 LOKI 算法的互补特性,当采用 LOKI 算法时是不安全的。

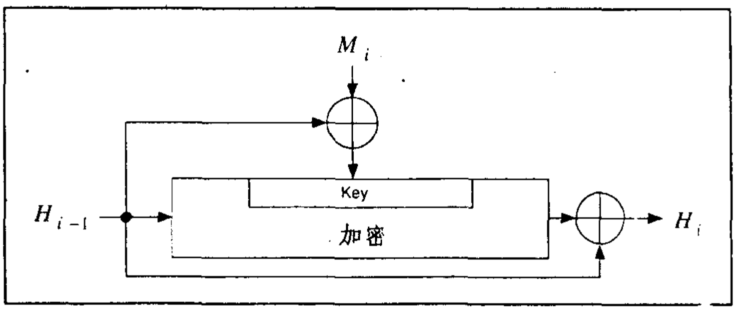


图 14.6 LOKI 单块哈希算法的结构

Miyaguchi 算法

这一算法(见图 14.7)产生长度与块长相同的哈希值。

$H_0 = I_H$ I_H 为随机初始值

$H_i = E_{H_{i-1}}(M_i) XOR H_{i-1} XOR M_i$

该算法还没有比蛮力攻击更容易的已知攻击方法。

其它块长与钥长相等的算法

B. Preneel 发现基于块密码、块长与钥长相等的 12 种算法,Davies-Meyer 算法与

Miyaguchi 算法是其中的二种，其它十种为：

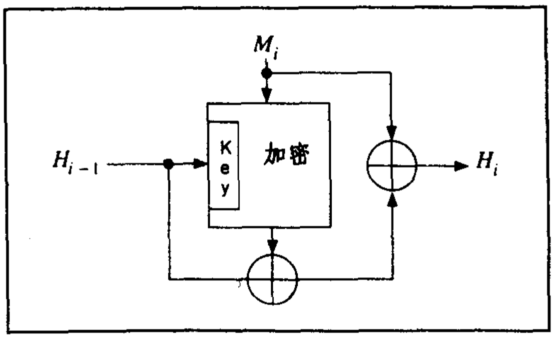


图 14.7 Miyaguchi 算法结构

$H_i = E_{Hi-1}(M_i)$ XOR M_i

$H_i = E_{Hi-1}(M_i$ XOR $H_{i-1})$ XOR M_i XOR H_{i-1}

$H_i = E_{Hi-1}(M_i$ XOR $H_{i-1})$ XOR M_i

$H_i = E_{Mi}(M_i$ XOR $H_{i-1})$ XOR M_i XOR H_{i-1}

$H_i = E_{Mi}(H_{i-1})$ XOR M_i XOR H_{i-1}

$H_i = E_{Mi}(M_i$ XOR $H_{i-1})$ XOR H_{i-1}

$H_i = E_{M_i XOR H_{i-1}}(M_i)$ XOR Xi

$H_i = E_{M_i XOR H_{i-1}}(H_{i-1})$XOR H_{i-1}

$H_i = E_{M_i XOR H_{i-1}}(M_i)$ XOR H_{i-1}

$H_i = E_{M_i XOR H_{i-1}}(H_{i-1})$ XOR X_i

所有都假设：

$H_0 = I_H$　　　　I_H 为随机初始值

修改的 Davies-Meyer 算法

Xuejia Lai 和 James Massey 修改了 Davies-Meyer 算法，而采用 IDEA 密码算法。IDEA 的块大小为 64 比特，钥长为 128 比特。算法为：

$H_0 = I$　　　　I 为随机初始值

$H_i = E_{H_{i-1}, M_i}(H_{i-1})$

这个函数把报文以 64 比特的块为单位哈希散列，在每轮中，IDEA 密码算法的密钥为前一轮的 64 比特哈希值与报文的 64 比特块。算法的输入为前一轮的哈希值，输出为当前轮的哈希值。注意与原 avies-Meyer 算法不同，输出不再与输入异或(见图 14.8)。

对这一算法还没有比蛮力攻击更容易的已知攻击方法。

Preneel-Bosselaers-Govaerts-Vandewalle 算法

该哈希函数生成的哈希值为密码块算法所用块长的二倍：64 比特的算法生成 128 比特的哈希值。

用 64 比特的块算法，这个算法生成二个 64 比特的哈希值 G_i 和 H_i，二者连接成为 128 比特的哈希值。二个与密钥等长的相邻报文块 L_i 和 R_i 一起哈希散列：

$G_0 = I_G$　　　　I_G 为随机初始值

$H_0 = I_H$ 　　　　I_H 为另一个随机初始值

$G_i = E_{li\ XOR\ H_{i-1}}(R_i\ XOR\ G_{i-1})\ XOR\ R_i\ XOR\ G_{i-1}\ XOR\ H_{i-1}$

$Hi = E_{li\ XOR\ Ri}(H_{i-1}\ XOR\ G_{i-1})\ XOR\ L_i\ XOR\ G_{i-1}\ XOR\ H_{i-1}$

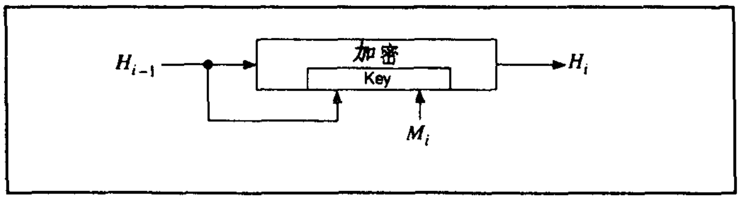


图 14.8 修改的 Davies－Meyer 算法结构

Xuejia Lai 说明了在某些情况下对这个算法的生日攻击稍稍可解，Preneel 和 D. Coppersmith 也对这个算法作了成功的攻击。

Quisquater-Girault 算法

这个算法也生成二倍块长的哈希值，与密钥等长的二块 Li 和 Ri 一起作哈希散列：

$G_0 = I_G$ 　　　　I_G 为随机初始值

$H_0 = I_H$ 　　　　I_H 为另一个随机初始值

$W_i = E_{li}(G_{i-1}\ XOR\ R_i)\ XOR\ R_i\ XOR\ H_{i-1}$

$G_i = E_{Ri}(W_i\ XOR\ L_i)\ XOR\ G_{i-1}\ XOR\ H_{i-1}\ XOR\ L_i$

$H_i = W_i\ XOR\ G_{i-1}$

有的情况下，对这个算法的生日攻击可在 2^{32} 复杂度内可解，而不是蛮力攻击所需的 2^{64}。

LOKI 双块算法

这个算法是 Quisquater-Girault 算法的修改，是为采用 LOKI 算法而专门设计的（如图 14.9），与密钥等长的二块 L_i 和 R_i 一起作哈希散列：

$G_0 = I_G$ 　　　　I_G 为随机初始值

$H_0 = I_H$ 　　　　I_H 为另一个随机初始值

$W_i = E_{L_i XOR\ G_{i-1}}(G_{i-1}\ XOR\ R_i)\ XOR\ R_i\ XOR\ H_{i-1}$

$G_i = E_{R_i XOR\ H_{i-1}}(W_i\ XOR\ L_i)\ XOR\ G_{i-1}\ XOR\ H_{i-1}\ XOR\ L_i$

$H_i = W_i\ XOR\ G_{i-1}$

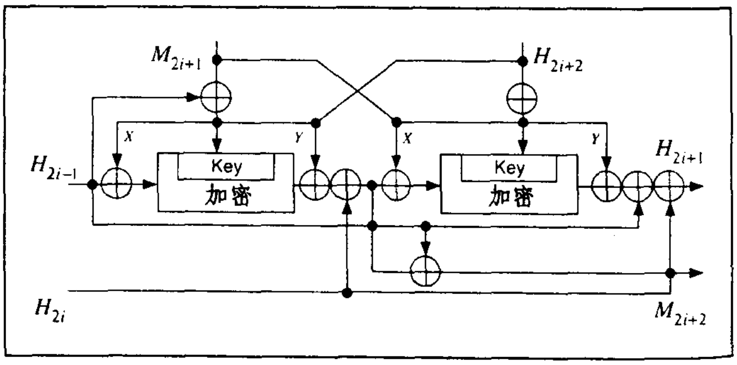


图 14.9 LOKI 双块哈希算法结构

有的情况下，生日攻击也稍稍可解。

串联的 Davies-Meyer 算法

由于 64 比特密钥的块密码算法内在限制，唯一的解决方法是采用 128 比特密钥的算法，如 IDEA。这个技术产生 128 比特的哈希值，二个修改后的 Davies-Meyer 函数串联工作（见图 14.10）。

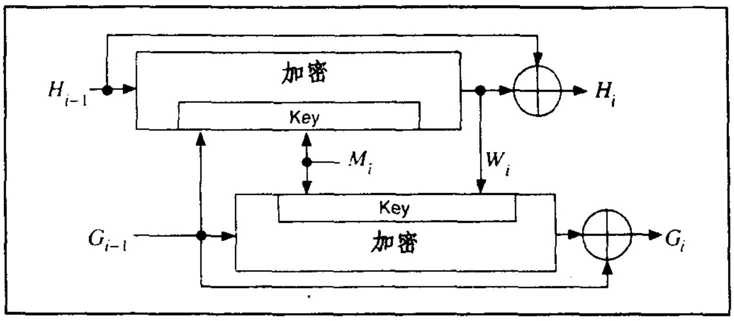


图 14.10　串级 Davies-Meyer 算法结构

这个算法产生二个 64 比特的哈希值 G_i 和 H_i，二者连接形成 128 比特的哈希值，块大小为 64 比特。

$G_0 = I_G$　　　　I_G 为随机初始值

$H_0 = I_H$　　　　I_H 为另一个随机初始值

$W_i = E_{G_{i-1},M_i}(H_{i-1})$

$G_i = G_{i-1}\ XOR\ E_{Mi,Wi}(G_{i-1})$

$H_i = W_i\ XOR\ H_{i-1}$

注意这个算法用二个 IDEA 加密来哈希散列每个 64 比特的报文块。

众人皆知，对于 128 比特的哈希函数，这个算法的安全性是理想的。假设对块算法没有比蛮力攻击更好的攻击方法，找到一个哈希散列到给定报文的报文需要 2^{128}次尝试，找到二个哈希值相同的随机报文需要 2^{64}次尝试。

并联的 Davies-Meyer 算法

这一技术也是由 Xuejia Lai 和 James Massey 设计的，二个修改的 Davies-Meyer 算法并排放置。它也用二个 64 比特的哈希值 G_i 和 H_i，二者连接形成 128bit 的哈希值，它也对每个 64 比特的报文块执行二个 IDEA 加密（见图 14.11）。

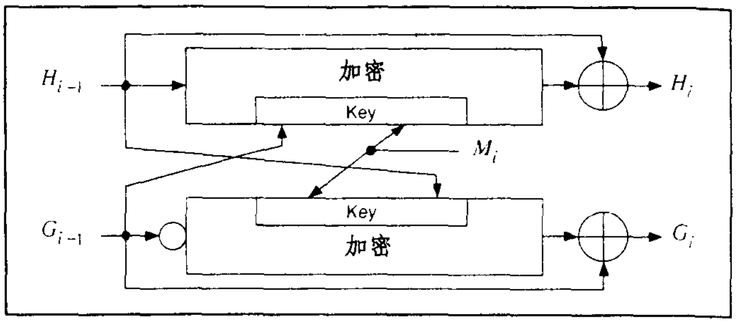


图 14.11　并联的 Davies-Meyer 算法结构

$G_0 = I_G$　　　　I_G 为随机初始值

$H_0 = I_H$　　　　I_H 为另一个随机初始值

$G_i = G_{i-1}\ XOR\ E_{mi,Hi-1}(\text{not}\ G_{i-1})$

$$H_i = H_{i-1} \text{ XOR } E_{G_{i-1}, M_i}(H_{i-1})$$

同前一算法一样，这个算法也有理想的安全性。

MDC-4 算法

MDC-4 算法是为 RIPE 项目而设计的，虽然理论上可采用任何加密算法，MDC-4 采用了 DES 作为块函数(处理一个 64 比特的报文块需要 4 次 DES 加密)，MDC-4 产生二倍于加密块长度的哈希值。

其它算法

大家仍然在研究设计快速而安全的哈希函数(每次加密处理一块)。R. Merkle 提出了一种用 DES 的算法，但是太慢，每次只处理 7 比特报文，然而如果增加一些对弱钥攻击的预防措施，它是安全的。另一个称为 Meyer-Schilling 的算法也有问题。

14.12 使用公钥算法

块链接方式的公钥加密算法可用作数字签名，如果抛弃私钥，攻破哈希值与没有私钥阅读报文同样困难。

下例这里说明如何将 RSA 变换为单向哈希函数。如果 M 为要哈希散列的报文，n 为二个素数 p 和 q 的乘积，e 为与(p－1)＊(q－1)互素的另一个大数，则哈希函数 H(M) 为：

$$H(M) = M^e \bmod n$$

更容易的方法是用一个强素数 p 作为模数：

$$H(M) = M^e \bmod p$$

攻破这个问题与求解 e 的离散对数一样难，如果 e 和 p 足够大，则单向哈希函数是安全的。主要的问题是这个算法太慢了。

14.13 密钥独立的单向哈希函数

密钥独立的单向哈希函数通常称为 MAC(Message Authentication Code，报文认证码)，MAC 具有前述单向哈希函数的一些特征，但也有一个密钥，只有用同一密钥才能检验哈希值。MAC 在不保密情况下作认证，非常有用。

块密码 MAC

生成密钥独立的单向哈希函数的最简单方法是用 CFB 方式的块算法加密报文，哈希值为最后一个加密块，再以 CFB 方式加密一次。这个算法在 ANSI X9.9 和 ISO 9797 说明。

RIPE-MAC 是 B. Preneel 设计并被 RIPE 项目采用，它基于 ISO 9797 并使用 DES 作为块密码函数。有二类 RIPE-MAC：一是使用一般的 DES，称为 RIPE-MAC1，另一个为提高安全性采用三重 DES，称为 RIPE-MAC3。

这个算法由三部分组成：首先，把报文长度扩展为 64 比特的倍数，其次，扩展的报文分成各为 64 比特的各块，在一个秘密密钥的控制之下，用一个加密的压缩函数把这些块哈希散列成 64 比特的块，这一步或者用 DES 或者用三重 DES。最后，对压缩函数的输出用不同的密钥作另一个基于 DES 的加密，这个密钥从用于压缩的密钥中导出。

单向哈希函数 MAC

单向哈希函数也可用作 MAC。假设 A 和 B 共享一个密钥 K，A 要送给 B 一个报文 M 的 MAC，A 把 M 和 K 连接并计算连接结果的单向哈希值，这个哈希值为 MAC。由于 B 知道 K，他可以复制 A 的结果，而不知道 K 的他人不能。

流密码 MAC

Xuejia Lai, R. Rueppll 和 J. Woollven 设计的一个真正精致的 MDC 算法采用流密码(见图 14.12)。一个密码安全的伪随机位发生器把报文流信号分离成二个子流,如果发生器的输出位 R_i 为 1,则当前报文位 m_i 发给第一个子流;如果 R_i 为 0,则当前报文位 m_i 发给第二个子流。二个子流分别输入给不同的反馈移位寄存器,MAC 的输出即为移位寄存器的最后状态。

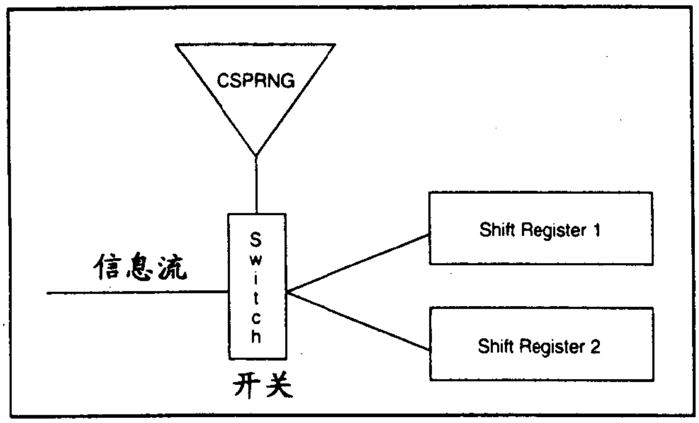


图 14.12 流密码 MAC

由于这个算法没有多重的迭代和漫长的一轮操作,它可以快速哈希散列报文。也许它是哈希散列高速数据流的最佳方法。

第 15 章 随机序列发生器和流密码

15.1 伪随机序列发生器

线性同余发生器

线性同余发生器是一个伪随机序列发生器，形式如下：

$$X_n=(aX_{n-1}+b)\bmod m$$

其中 X_n 为序列中的第 n 个数，X_{n-1} 为序列的前一个数，变量 a，b 和 m 是常量：a 为系数，b 为增量，m 为模数。种子为 X 的初值。

发生器的周期不大于 m，如果适当地选取 a，b 和 m，这个发生器会是周期为 m 的最长发生器（例如，m 应选择为素数）。

表 15.1 给出线性同余发生器的一些推荐值，它们都会生成最大长度发生器，更为重要的是，它们通过 Knuth 的 2 至 6 维的随机谱测试。按不超过确定字长的最大乘积而组织成下表，以不溢出给定字长的最大乘积而组织。

表 15.1 线性同余发生器的常量

溢出点	a	b	m	溢出点	a	b	m
2^{20}	106	1283	6075	2^{28}	1277	24749	117128
2^{21}	211	1663	7875		741	66037	312500
2^{22}	421	1663	7875		2041	25673	121500
2^{23}	430	2531	11979	2^{29}	2311	25367	120050
	936	1399	6655		1807	45289	214326
	1366	1283	6075		1597	51749	244944
2^{24}	171	11213	53125		1861	49297	233280
	859	2531	11979		2661	36979	175000
	419	6173	29282		4081	25673	121500
	967	3041	14406		3661	30809	145800
2^{25}	141	28411	134456	2^{30}	3877	29573	139968
	625	6571	31104		3613	45289	214326
	1541	2957	14000		1366	150889	714025
	1741	2731	12960	2^{31}	8121	28411	134456
	1291	4621	21870		4561	51349	243000
	205	29573	139968		7141	54773	259200
2^{26}	421	17117	81000	2^{32}	9301	49297	233280
	1255	6173	29282		4096	150889	714025
	281	28411	134456	2^{33}	2416	374441	1771875
2^{27}	1093	18257	86436	2^{34}	17221	107839	510300
	421	54773	259200		36261	66037	312500
	1021	24631	116640	2^{35}	84589	45989	217728
	1021	25673	121500				

线性同余发生器的优点是快速，每位只需少量的操作。

但是，由于线性同余发生器是可预测的，因此不能用于流密码密码术。线性同余发生器首次被 J. Boyar 攻破，她也攻破了平方发生器：

$$X_n=(aX_{n-1}^2+bX_{n-1}+c)\bmod m$$

和立方发生器：

$$X_n=(aX_{n-1}^3+bX_{n-1}^2+cX_{n-1}+d)\bmod m$$

其他研究人员把 Boyar 的工作推广到攻破任何多项式同余发生器，并且 Stern 和 Boyar 说明了即使不知道整个序列的情况下如何攻破线性同余发生器。

线性同余发生器对于非密码术应用（例如仿真）还是有用的，它们在大多数经验测试方面具有良好的统计特性。

组合线性同余发生器

Wichmann 和 Hill，以及后来的 P. L'Ecuyer 分析了组合线性同余发生器，他们的结论是这类发生器没有更高的密码安全性，但是周期更长，并在一些随机性测试方面更好。

L'Ecuyer 给出了用于 32 位计算机的发生器程序：

```
static long s1=1;     /^"long"must be 32 bits long. * /
static long s2=1;
#define MODMULT(a,b,c,m,s)q=s/a;s=b * (s-a * q)-c * q;if(s<0)s+=m;
/ * MODMULT(a,b,c,m,s)computes s * b mod m,provided that m=a * b+c
and 0<=c<m. * /
/ * combinedLCG returns a pseudorandom real value in the range
(0,1). It combines linear congruential generators with
periods of 2 * * 31-85 and 2 * * 31-249,and has a period that
is the product of these two prime numbers.  * /
double combinedLCG(void)
{
  long q,z;
  MODMULT(53668,40014,12211,2147483563L,s1)
  MODMULT(52774,40692, 3791,2147483399L,s2)
  z=s1-s2;
  if(z<1)z+=2147483562;
  return z * 4.656613e-10;
}
/ * In general,call initLCG before using combinedLCG. * /
void initLCG(long InitS1,long InitS2)
{
  s1=InitS1;
  s2=InitS2;
}
```

这个发生器可接受在 $-2^{31}+85$ 至 $2^{31}-85$ 之间计算机能表示的所有整数，变量 s1 和 s2

是全程变量，保存发生器的当前状态，必须在第一次调用之前初始化。变量 s1 需要一个 1 到 2147483562 之间的初值，变量 s2 需要一个 1 到 2147483398 之间的初值，这个发生器的周期在 10^{18}附近。

如果只有 16 位的计算机，则可采用如下发生器：

```
static long s1=1;          /* A"int"must be 16 bits long. */
static long s2=1;
static long s3=1;
#define MODMULT(a,b,c,m,s)q=s/a;s=b*(s-a*q)-c*q;if(s<0)s+=m;

/* combinedLCG returns a pseudorandom real value in the range
(0,1). It combines linear congruential generators with
periods of 2**15-405,2**15-1041,and 2**15-1111,and has a
period that is the product of these three prime numbers. */
double combinedLCG(void)
{
  long q,z;
  MODMULT(206,157,21,32363,s1)
  MODMULT(217,146,45,31727,s2)
  MODMULT(222,142,133,31657,s3)
  z=s1-s2;
  if(z>706)z-=32362;
  z+=s3;
  if(z<1)z+=32362;
  return z*3.0899e-5;
}

/* In general,call initLCG before using combinedLCG. */
void initLCG(int InitS1,int InitS2,int InitS3)
{
  s1=InitS1;
  s2=InitS2;
  s3=InitS3;
}
```

这个发生器可接受在－32363 至 32363 之间计算机能表示的所有整数，变量 s1、s2 和 s3 是全程变量，保存发生器的当前状态，必须在第一次调用之前初始化。变量 s1 需要一个 1 到 32362 之间的初值，变量 s2 需要一个 1 到 31762 之间的初值，变量 s3 需要一个 1 到 31656 之间的初值，这个发生器的周期为 1.6×10^{13}。

这二个发生器中的常量 b 为 0。

线性反馈移位寄存器

线性反馈移位寄存器(LFSR),由一个移位寄存器和一个分支序列(tap sequence)组成(见图 15.1),移位寄存器是位序列,每次需要一位(由于这个方法往往用硬件实现,有时也称之为时钟脉冲),移位寄存器中的所有位右移并且在 LFSR 的最低位输出,新的最高位与分支序列同时由寄存器中的其它位异或而得。

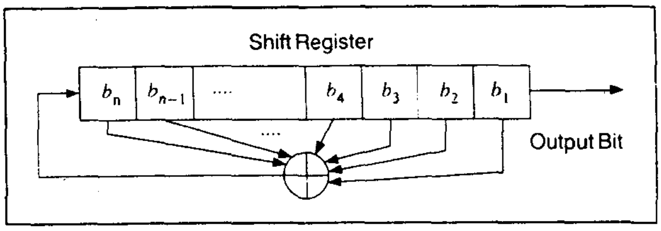


图 15.1 线性反馈移位寄存器

理论上,n 比特的 LFSR 在重复之前可生成 2^n-1 比特的伪随机序列,为此,移位寄存器必须循环所有 2^n-1 个内部状态(2^n-1 而不是 2^n 是由于全 0 填充的移位寄存器会使 LFSR 输出无终止的 0 流,这是没有用的),只有某种分支序列的 LFSR 才能循环所有 2^n-1 个内部状态,这些 LFSR 称为最大长度的 LFSR。

例如,在第 1 和第 4 位分支的 4 比特 LFSR(见图 15.2),以 1111 初始化,在重复之前生成下列序列的内部状态:

1111
0111
1011
0101
1010
1101
0110
0011
1001
0100
0010
0001
1000
1100
1110

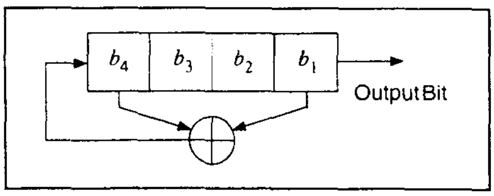


图 15.2 4 比特的 LFSR

每一状态的最低位构成输出序列为:

111101011001000…

为使 LFSR 成为最大长度的 LFSR，分支序列的元素构成的多项式加上常量 1 必须是模 2 的本原多项式(primitive polynomial)。n 阶的本原多项式是除尽 $x^{2^n-1}+1$ 的不可约多项式，但不是对任意 d 除尽 2^n-1 的 x^d+1。

表 15.2 是一些模 2 的本原多项式，例如，(32，7，5，3，2，1，0)表示下面多项式是模 2 的本原多项式：

$$x^{32}+x^7+x^5+x^3+x^2+x^1+1$$

把它转化为最大长度的 LFSR 是容易的，第一个数是阶以及 LFSR 的长度，最后一个数总是 0 可以省略，除 0 之外的所有数描述分支序列。

表 15.2　一些模 2 的本原多项式

(1,0)	(2,1,0)	(3,1,0)	(4,1,0)
(5,2,0)	(6,1,0)	(7,1,0)	(7,3,0)
(8,4,3,2,0)	(9,4,0)	(10,3,0)	(11,2,0)
(12,6,4,1,0)	(13,4,3,1,0)	(14,5,3,1,0)	(15,1,0)
(16,5,3,2,0)	(17,3,0)	(17,5,0)	(17,6,0)
(18,7,0)	(18,5,2,1,0)	(19,5,2,1,0)	(20,3,0)
(21,2,0)	(22,1,0)	(23,5,0)	(24,4,3,1,0)
(25,3,0)	(26,6,2,1,0)	(27,5,2,1,0)	(28,3,0)
(29,2,0)	(30,6,4,1,0)	(31,3,0)	(31,6,0)
(31,7,0)	(31,13,0)	(32,7,6,2,0)	(33,13,0)
(33,16,4,1,0)	(34,8,4,3,0)	(34,7,6,5,4,1,0)	(35,2,0)
(36,11,0)	(36,6,5,4,2,1,0)	(37,6,4,1,0)	(37,5,4,3,2,1,0)
(38,6,5,1,0)	(39,4,0)	(40,5,4,3,0)	(41,3,0)
(42,7,4,3,0)	(42,5,4,3,2,1,0)	(43,6,4,3,0)	(44,6,5,2,0)
(45,4,3,1,0)	(46,8,7,6,0)	(46,8,5,3,2,1,0)	(47,5,0)
(48,9,7,4,0)	(48,7,5,4,2,1,0)	(49,9,0)	(49,6,5,4,0)
(50,4,3,2,0)	(51,6,3,1,0)	(52,3,0)	(53,6,2,1,0)
(54,8,6,3,0)	(54,6,5,4,3,2,0)	(55,24,0)	(55,6,2,1,0)
(56,7,4,2,0)	(57,7,0)	(57,5,3,2,0)	(58,19,0)
(58,6,5,1,0)	(59,7,4,2,0)	(59,6,5,4,3,1,0)	(60,1,0)
(61,5,2,1,0)	(62,6,5,3,0)	(63,1,0)	(64,4,3,1,0)
(65,18,0)	(65,4,3,1,0)	(66,9,8,6,0)	(66,8,6,5,3,2,0)
(67,5,2,1,0)	(68,9,0)	(68,7,5,1,0)	(70,5,3,1,0)
(70,5,3,1,0)	(71,6,0)	(71,5,3,1,0)	(72,10,9,3,0)
(72,6,4,3,2,1,0)	(73,25,0)	(73,4,3,2,0)	(74,7,4,3,0)
(75,6,3,1,0)	(76,5,4,2,0)	(77,6,5,2,0)	(78,7,2,1,0)
(79,9,0)	(79,4,3,2,0)	(80,9,4,2,0)	(80,7,5,3,2,1,0)
(81,4,0)	(82,9,6,4,0)	(82,8,7,6,1,0)	(83,7,4,2,0)
(84,13,0)	(84,8,7,5,3,1,0)	(85,8,2,1,0)	(86,6,5,2,0)
(87,13,0)	(87,7,5,1,0)	(88,11,9,8,0)	(88,8,5,4,3,1,0)
(89,38,0)	(89,51,0)	(89,6,5,3,0)	(90,5,3,2,0)
(91,8,5,1,0)	(91,7,6,5,3,2,0)	(92,6,5,2,0)	(93,2,0)
(94,21,0)	(94,6,5,1,0)	(95,11,0)	(95,6,5,4,2,1,0)
(96,10,9,6,0)	(96,7,6,4,3,2,0)	(97,6,0)	(98,11,0)
(98,7,4,3,1,0)	(99,7,5,4,0)	(100,37,0)	(100,8,7,2,0)
(101,7,6,1,0)	(102,6,5,3,0)	(103,9,0)	(104,11,10,1,0)
(105,16,0)	(106,15,0)	(107,9,7,4,0)	(108,31,0)
(109,5,4,2,0)	(110,6,4,1,0)	(111,10,0)	(111,49,0)
(112,11,6,4,0)	(113,9,0)	(113,15,0)	(113,30,0)
(114,11,2,1,0)	(115,8,7,5,0)	(116,6,5,2,0)	(117,5,2,1,0)

续上表

(118,33,0)	(119,45,0)	(119,8,0)	(120,9,6,2,0)
(121,18,0)	(122,6,2,1,0)	(123,2,0)	(124,37,0)
(125,7,6,5,0)	(126,7,4,2,0)	(127,1,0)	(127,7,0)
(127,15,0)	(127,30,0)	(127,63,0)	(128,7,2,1,0)
(129,5,0)	(130,3,0)	(131,8,3,2,0)	(132,29,0)
(133,9,8,2,0)	(134,57,0)	(135,11,0)	(135,16,0)
(135,22,0)	(136,8,3,2,0)	(137,21,0)	(138,8,7,1,0)
(139,8,5,3,0)	(140,29,0)	(141,13,6,1,0)	(142,21,0)
(143,5,3,2,0)	(144,7,4,2,0)	(145,52,0)	(145,69,0)
(146,5,3,2,0)	(147,11,4,2,0)	(148,27,0)	(149,10,9,7,0)
(150,53,0)	(151,3,0)	(151,9,0)	(151,15,0)
(151,31,0)	(151,39,0)	(151,43,0)	(151,46,0)
(151,51,0)	(151,63,0)	(151,66,0)	(151,67,0)
(151,70,0)	(152,6,3,2,0)	(153,1,0)	(153,8,0)
(154,9,5,1,0)	(155,7,5,4,0)	(156,9,5,3,0)	(157,6,5,2,0)
(158,8,6,5,0)	(159,31,0)	(159,34,0)	(159,40,0)
(160,5,3,2,0)	(161,18,0)	(161,39,0)	(161,60,0)
(162,8,7,4,0)	(163,7,6,3,0)	(164,12,6,5,0)	(165,9,8,3,0)
(166,10,3,2,0)	(167,6,0)	(170,23,0)	(172,2,0)
(174,13,0)	(175,6,0)	(175,16,0)	(175,18,0)
(175,57,0)	(177,8,0)	(177,22,0)	(177,88,0)
(178,87,0)	(183,56,0)	(194,87,0)	(198,65,0)
(201,14,0)	(201,17,0)	(201,59,0)	(201,79,0)
(202,55,0)	(207,43,0)	(212,105,0)	(218,11,0)
(218,15,0)	(218,71,0)	(218,83,0)	(225,32,0)
(225,74,0)	(225,88,0)	(225,97,0)	(225,109,0)
(231,26,0)	(231,34,0)	(234,31,0)	(234,103,0)
(236,5,0)	(250,103,0)	(255,52,0)	(255,56,0)
(255,82,0)	(258,83,0)	(266,47,0)	(270,133,0)
(282,35,0)	(282,43,0)	(286,69,0)	(286,73,0)
(294,61,0)	(322,67,0)	(333,2,0)	(350,53,0)
(366,29,0)	(378,43,0)	(378,107,0)	(390,89,0)
(462,73,0)	(521,32,0)	(521,48,0)	(521,158,0)
(521,168,0)	(607,105,0)	(607,147,0)	(607,273,0)
(1279,216,0)	(1279,418,0)	(2281,715,0)	(2281,915,0)
(2281,1029,0)	(3217,67,0)	(3217,576,0)	(4423,271,0)
(9689,84,0)			

在本例中,(32,7,5,3,2,1,0)表示如果用一个 32 比特的移位寄存器,并把第 32、第 7、第 5、第 3、第 2 和第 1 位异或在一起而生成新的位(见图 15.3),则产生的 LFSR 为最大长度的 LFSR:它在重复之前循环 $2^{32}-1$ 个值。

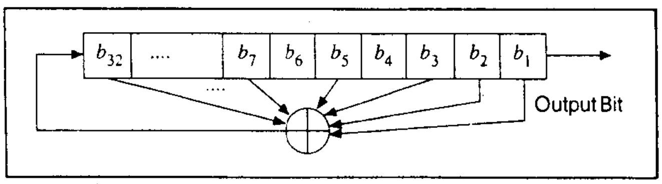


图 15.3　32 比特的最大长度 LFSR

这个 LFSR 的 C 代码为：

```
int LFSR()
    {
    static unsigned long ShiftRegister=1;/ * Anything but 0 * /
    ShiftRegister=((((ShiftRegister>>7)
                ^ (ShiftRegister>>5)
                ^ (ShiftRegister>>3)
                ^ (ShiftRegister>>2)
                ^ (ShiftRegister>>1)
                ^ (ShiftRegister)
                &0x00000001)
                <31)
                |(ShiftRegister>>1);
    return ShiftRegister & 0x00000001;
    }
```

当移位寄存器比计算机的字长更长时，代码稍稍复杂。

注意表15.2中所有本原多项式的系数个数都为奇数，并且由于如果 p(x)为本原元，则 x^n p(1/x)也是本原元，因此表中每一项实际上是二个本原多项式。

例如，如果(a,b,0)为本原元，则(a,a－b,0)也是本原元；如果(a,b,c,d,0)为本原元，则(a,a－d,a－c,a－b,0)也是本原元。数学表示为：

如果 x^a+x^b+1 为本原元，则 $x^a+x^{a-b}+1$ 也是本原元

如果 $x^a+x^b+x^c+x^d+1$ 为本原元，则 $x^a+x^{a-d}+x^{a-c}+x^{a-b}+1$ 也是本原元本原三项式由于移位寄存器中只有二位需要异或，故特别有用，然而，攻击它也容易得多。

LFSR 自身是比较合适的随机序列发生器，但具有一些非随机特性，顺序的位是线性的，加密时无用。长度为 n 的 LFSR，内部状态为发生器的前 n 比特输出。即使不知道反馈机制，只需取发生器的2n 比特输出就可确定。

从序列的顺序位生成的大随机数是高度相关的，并且对某些应用场合并不随机，虽然如此，LFSR 往往在加密算法中用作构造块。

修改的 LFSR

可以对 LFSR 的反馈机制作修改，所产生的发生器虽无更好的密码功能，但仍然是最大长度并更易于软件实现。

修改后的 LFSR 不再采用分支序列中的各位来生成新的最高位，而是让分支序列中的每一位都与发生器的输出异或并由异或结果替换各位，发生器的输出成为新的最高位(见图15.4)。

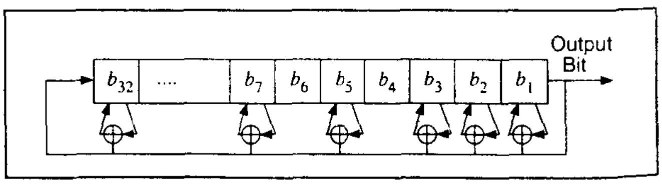


图15.4　修改的32比特 LFSR

其C实现为：

```
#define mask 0x80000057
static unsigned long ShiftRegister=1;
void seed_LFSR(unsigned long seed)
{
if(seed==0)/*avoid calamity */
seed=1;
ShiftRegister=seed;
}
int modified_LFSR(void)
{
if(ShiftRegister & 0x00000001)
{
ShiftRegister=(ShiftRegister ^ (mask>>1))|0x8000000;
return 1;
}
else
{
    ShiftRegister>>=1;
    return 0;
  }
}
```

其优点是所有异或可作为单一操作完成，如果分支序列为模2的本原多项式，则序列发生器是最大长度的。

15.2 流密码

密码安全的伪随机序列发生器(CSPRSG)用于流密码时十分完美，这些发生器的输出与真正随机的位发生器难以区分(至少在合理的时间内)，只需将发生器的输出与明文流异或就可得到良好的流密码。

R. Rueppel认为构造CSPRSG有四种实用方法：信息论方法、系统论方法、复杂性理论方法和随机化方法。这些方法的区别在于性能和密码分析者的机遇不同和密码成功的定义以及安全性的考虑。这个领域上大多数的研究只是理论上的，但也有一些好的密码安全的发生器。

15.2.1 系统论方法

在流密码的系统论方法中，密码编制人员设计具有安全性可测特征的密钥流发生器，这些特征包括周期、位模式的分布、线性复杂度等等，密码编制人员也研究对这些发生器的各种密码分析技术，并保证这些发生器对这些攻击是免疫的。

过去的数年中，已经取得针对这个方法的一些密钥流发生器的设计准则，Rueppel描述如下：

·长周期，没有重复

·线性复杂度准则：大的线性复杂度，线性复杂度概貌，局部线性复杂度等（发生器的线性复杂度乃是形成发生器输出的最短 LFSR 的长度，线性复杂度是发生器随机性的尺度）

·统计标准例如理想 k 维分布

·混乱：密钥流的每一位必须是所有或大部分密钥位的复杂变换

·扩散：子结构中的冗余度必须分散到大范围的统计量中

·布尔函数的非线性准则，如 m 阶相关免疫、与线性函数的距离、雪崩效应等。

这些设计准则并不只是针对由系统论方法设计的流密码，它们对所有的流密码都正确，对所有块密码更加正确。对系统论方法来讲唯一的不同是流密码设计成直接满足这些目标。

这些密码体制的主要问题是无法证明其安全性，从未证明这些设计准则对于安全性是否必要和充分。满足所有设计准则的某个密钥流发生器可能是不安全的，而另一个也许是安全的。

另一方面，攻破每个密钥流发生器是一个与密码破译不同的问题，如果有足够多不同的发生器，密码分析者不值得逐个试图攻破，也许采用别的方法更好，如对大数因子分解或者计算离散对数。

最后，流密码与本书中的所有对称块算法没有什么不同：无安全性证明；存在着许多攻击方法；能抵抗这些攻击并不保证能抵抗新的攻击；这个领域发展迅速，目前大多数军事加密系统都基于 LFSR。

Gaffe 发生器

这个密钥流发生器采用三个 LFSR，以非线性方式组合（见图15.5），二个 LFSR 输入一个转换开关，第三个 LFSR 控制转换开关的输出。如果 a_1，a_2和 a_3为三个 LFSR 的输出，Gaffe 发生器的输出为：

$$b = a_2 a_1 \text{XOR} (\text{NOT } a_2) a_3$$

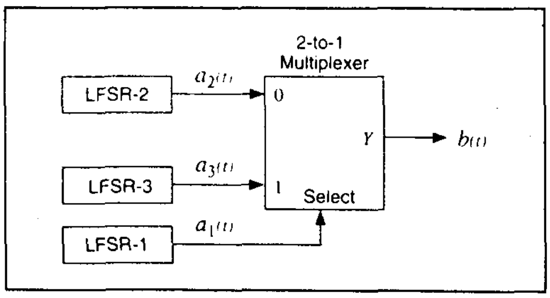


图15.5 非线性方式组合

如果三个 LFSR 的本原反馈多项式的阶各为 n_1，n_2和 n_3，则发生器的线性复杂度为：

$$(n_1+1)n_2+n_1n_3$$

发生器的周期为三个发生器周期的最小公倍数，假如三个本原反馈多项式的阶互素，发生器的周期为三个 LFSR 周期的乘积。

Gaffe 发生器的密码性能是很脆弱的，从概率上看发生器的输出75％时间等于 LFSR-1的输出，或者，75％的时间等于 LFSR-2的输出，因此可以相当容易地攻破这种密钥流发生器。例如，如果每个本原多项式只有三项，并且多项式的最高阶为 n，则只需取输出序列的37n 比特

长的一段就可重构所有三个 LFSR 的内部状态。

Pless 发生器

根据 J-K 触发器的性能设计成该发生器，它由驱动4个 J-K 触发器的8个 LFSR 构成，每个触发器作为2个 LFSR 的非线性组合器。为避免触发器的输出与源数据和下一个输出值等同，4个触发器各取1/4并交错形成最后的密钥流。

通过分别攻击每个触发器，完成该算法的密码破译，另外，组合的 J-K 触发器的密码性能是很脆弱的，这类发生器会被相关攻击攻破。

多路开关发生器

这个方法用一个多路开关来组合二个 LFSR，多路开关的输出位由 LFSR-1控制，并从 LFSR-2中选取。另外还有一个函数把 LFSR-2的输出映射为多路开关的输入（见图15.6）。

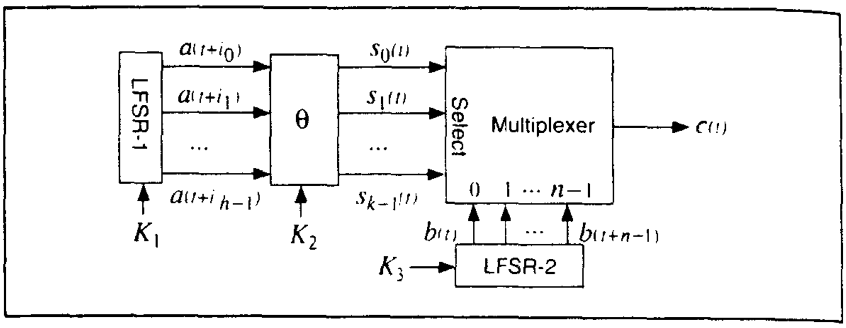


图15.6　多路开关发生器

密钥为二个 LFSR 和映射函数的内部状态。虽然这个发生器具有良好的统计特性，但被 R. Anderson 的密码分析技术攻破了。

Beth-Piper“停-走”式发生器

这个发生器（见图15.7）用一个 LFSR 的输出来控制另一个 LFSR 的时钟，LFSR-2的时钟输入受 LFSR-1的输出控制，这样只有当 t-1时刻 LFSR-1的输出为1时 LFSR-2才能在 t 时刻改变状态。

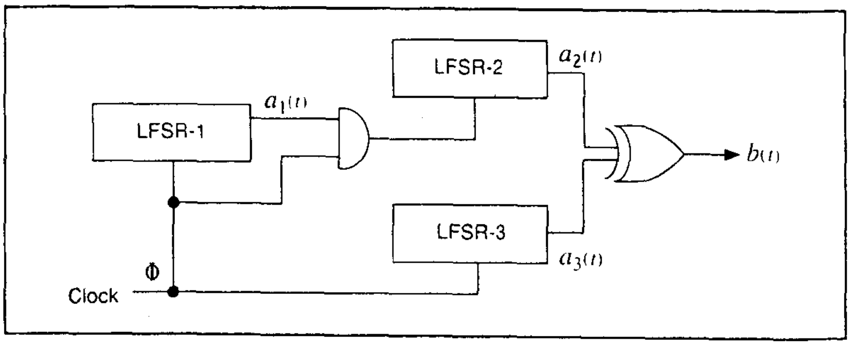


图15.7　Beth-Piper“停-走”式发生器

假设三个 LFSR 的长度互素并且反馈序列为本原多项式，这个发生器的线性复杂度为：

$$(2^{n_1}-1)n_2+n_3$$

然而，其安全性与 Gaffe 发生器一样差。

Gollmann 级联发生器

Gollmann 级联发生器是 Beth-Piper 发生器的增强版(见图15.8),它由一串 LFSR 构成,每个 LFSR 的时钟由前一个 LFSR 控制,如果 t－1时刻 LFSR-1的输出为1,则 LFSR-2在 t 时刻阶跃;如果 t－1时刻 LFSR-2的输出为1,则 LFSR-3在 t 时刻阶跃等等,最后一个 LFSR 的输出为发生器的输出。如果所有 LFSR 的长度都为1,则 n 个 LFSR 构成的系统的线性复杂度为:

$$1(2^1-1)^{n-1}$$

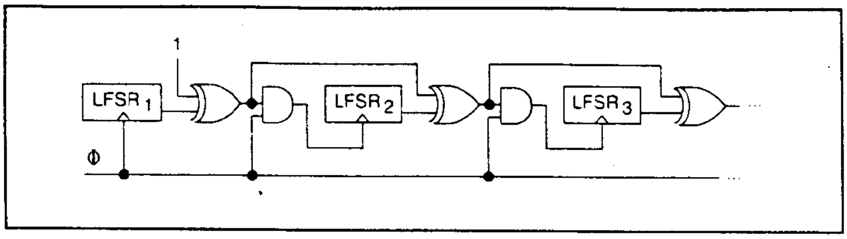


图15.8 Gollmann 级联发生器

交替的"停-走"式发生器

这个发生器用三个不同长度的 LFSR,当 LFSR-1的输出为1时 LFSR-2的时钟才有效,而当 LFSR-1的输出为0时 LFSR-3的时钟才有效,发生器的输出为 LFSR-2和 LFSR-3的输出的异或(见图15.9)。

这个发生器的周期很长并且具有很大的线性复杂度。

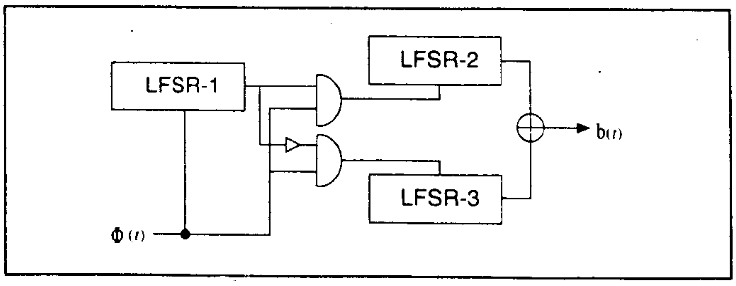


图15.9 交替的"停-走"式发生器

Bilateral"停-走"式发生器

这个发生器用二个长度都为1的LFSR(见图15.10),发生器的输出为各个LFSR输出的

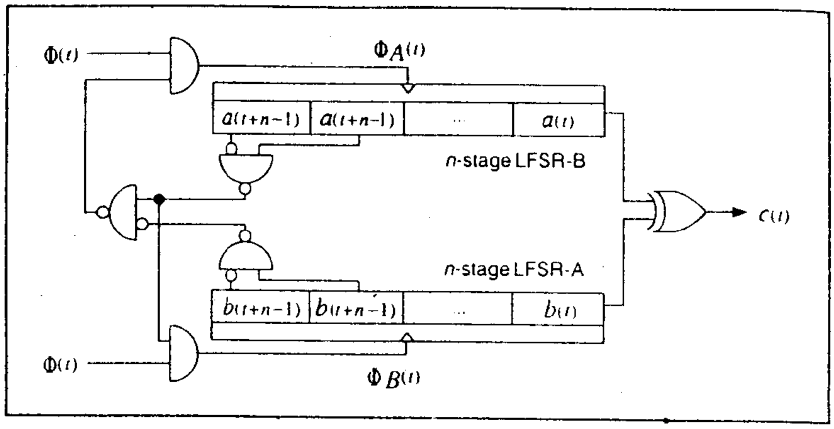


图15.10 Bilateral"停-走"式发生器

异或，如果 LFSR-A 在 t－1时刻的输出为0并且在 t－2时刻的输出为1，则 LFSR-B 在 t 时刻不阶跃，相反，如果 LFSR-B 在 t－1时刻的输出为0并且在 t－2时刻的输出为1，以及如果 LFSR-B 在 t 时刻阶跃，则 LFSR-A 在 t 时刻不阶跃。

这个发生器的线性复杂度基本上等于周期。

门限发生器

这个发生器试图用可变数目的 LFSR 达到前面发生器的安全性，其思想是如果采用很多 LFSR，则攻破密码就越难。

如图15.11所示，发生器取大量 LFSR 的输出（奇数个）。为使周期最大，要保证所有 LFSR 的长度互素并且所有反馈多项式都是本原元。如果半数以上输出为1，则发生器的输出为1，如果半数以上输出为0，则发生器的输出为0。

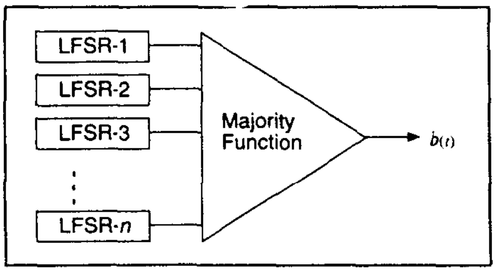


图15.11　门限发生器

三个 LFSR 的发生器的输出为：

$$b=a_1a_2 \text{ XOR } a_1a_3 \text{ XOR } a_2a_3$$

这非常类似于 Gaffe 发生器，只是它的线性复杂度很大：

$$L_1L_2+L_1L_3+L_2L_3$$

其中 L_1、L_2和 L_3 分别是第1、第2和第3个 LFSR 的长度。

这个发生器并不太好，发生器的每一位输出都产生一些有关 LFSR 状态的信息。

Self-Decimated 发生器

这类发生器自身控制其时钟，已经提出了二个这类发生器，一个是由 Rueppel 提出的（见图15.12），另一个是由 Chambers 和 Gollmann 提出的（见图15.13）。在 Rueppel 的发生器中，当LFSR的输出为0时，LFSR输入d个时钟，当LFSR的输出为1时，LFSR输入k个时钟。

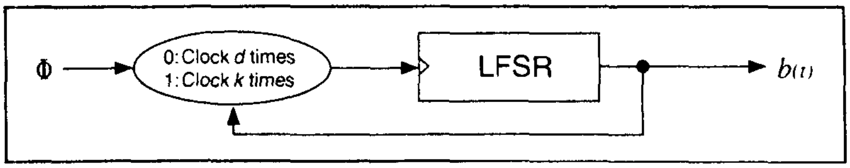


图15.12　Rueppel 的发生器

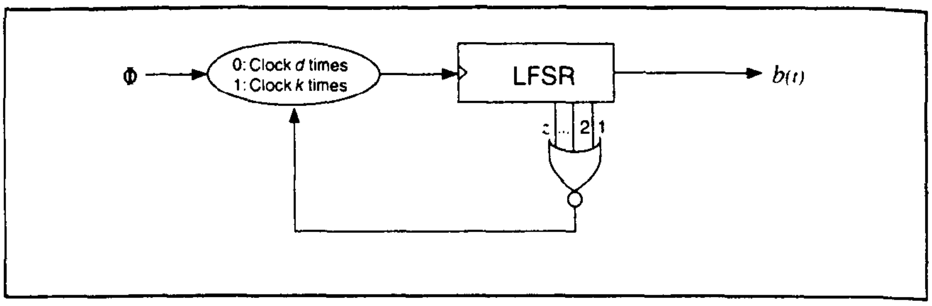


图15.13　Chambers 和 Gollmann 的发生器

Chambers 和 Gollmann 的发生器更为复杂，但思想是一样的。然而，这二个发生器都不安全。

多速度内积(multispeed inner-product)**发生器**

由 Massey 和 Ruppel 提出的这个发生器用二个不同速度时钟的 LFSR（见图15.14)，LFSR-2的时钟速度为 LFSR-1的 d 倍，二个 LFSR 的对应各位相与并异或形成发生器的最后输出。

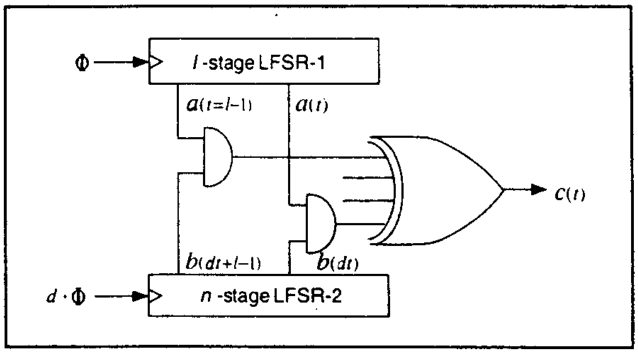


图15.14　多速度内积发生器

尽管这个发生器的线性复杂度很高并且处理良好的统计特性，但还是不安全。如果 LFSR-1的长度为 l，LFSR-2的长度为 n，二者的速度倍数为 d，则发生器的内部状态可从长度为

$$l+n+\log_2 d$$

的输出序列中恢复。

细胞自动机发生器

S. Wolfram 提出采用诸如 CSPRNG 之类的一维细胞自动机，发生器由一维的位串 $a_1, a_2, \cdots, a_k, \cdots, a_n$ 以及一个更新函数：

$$a_k' = a_{k-1} \text{XOR} (a_k \text{ OR } a_{k+1})$$

构成，位是任意从 a_k 中抽取一位。

这个发生器似乎相当随机，然而，对这些发生器存在一个已知明文攻击，这种攻击在 PC 机上多达500比特的 n 上进行。另外，P. Bardell 证明了细胞自动机的输出与线性反馈移位寄存器的输出完全一致，因此也并不安全。

1/p 发生器

如果这个发生器在 t 时刻的内部状态为 x_t，则

$$x_{t+1} = bx_t \bmod p$$

发生器的输出为

$$x_t \text{ div } p \qquad \text{(div 为截尾整除)}$$

的最低位。

为使周期最大，常量 p 应为素数并且常量 b 应为模 p 的本原根，这个发生器也不安全。

累加发生器

由 R. Rueppel 提出的这个发生器简单地把二个 LFSR 的输出通过一个相加器（带阵列）累加，这个操作是高度非线性的，直到80年代末它还是安全的，最近被攻破了。

CRYPT(1)

原始 UNIX 加密算法是个基于与 Enigma 相同技术的流密码，它是一个256元素、带反映器(reflector)的单回转子替代密码，反映器和回转子都由密钥生成。这个算法比二战时德国的

Enigma 要简单得多，一位训练有素的密码分析者轻易即可攻破。一个称为 Crypt Breakers Workbench(CBW)的公开 UNIX 程序可用于攻破由 crypt(1)加密的文件。

其它方法

RC4是由 RSA 数据安全公司开发的流密码，另外也还有基于背包问题的发生器。

15.2.2 复杂性理论方法

在这种方法中，密码编制人员试图采用复杂性理论来证明其发生器是安全的，因此，这种发生器趋于更复杂化，它建筑在公钥密码术中的那些"难"问题之上，并类似于公钥算法，速度慢并且不方便。

Shamir 伪随机数发生器

Shamir 采用 RSA 算法作为伪随机数发生器。

Blum-Micali 发生器

这个发生器的安全性在于计算离散对数的难度。设 a 为素数，p 为奇素数，随机种子 x_0开始处理过程：

$$x_{i+1}=ax_i \bmod p$$

如果 $x<(p-1)/2$则发生器输出为1，否则为0。

如果 p 足够大，以致于计算模 p 的离散对数不可行，则这个发生器是安全的。

RSA

这个 RSA 发生器初始参数为二个大素数之积 N 和一个随机种子 x_0，其中 x_0小于 N。

$$x_{i+1}=x_i^e \bmod N$$

发生器的输出为 x_i 的最低位。这个发生器的安全性基于攻破 RSA 的难度，如果 N 足够大，则发生器是安全的。

Blum Blum Shub

目前最简单也最有效的发生器称为 BBS(Blum Blum Shub)发生器，取名源自发明者，它有时也称为平方余数发生器。

BBS 发生器的理论是执行模 n 的平方余数(见9.3节)，其处理过程为：

首先找到模4同余于3的二个大素数 p 和 q，其乘积 n=pq 为 Blum 整数，选择另一个与 n 互素的随机整数 x，计算：

$$x_0=x^2 \bmod n$$

这是发生器的种子。

第 i 个伪随机位是 x_i 的最低位，其中 x_i 为：

$$x_i=x_{i-1}^2 \bmod n$$

这个发生器最迷人的特性是无需迭代所有 i-1比特即可得到第 i 比特。如果知道 p 和 q，就可直接计算第 i 比特：

b_i 为 x_i 的最低位，其中 x_i 为

$$x_i=x_0^{2^{\wedge}i \bmod ((p-1)(q-1))} \bmod n$$

这一特性意味着可以把这种密码坚固的伪随机位发生器用作随机访问文件的流密码系统。

这个方法的安全性在于对 n 作因子分解的难度。可以公开 n，这样人人都可以用这个发生器生成各位，然而，除非密码分析者能够对 n 作因子分解，他们就无法预测发生器的输出。

更为坚固的是，BBS 发生器是左不可预测和右不可预测的，即给定由发生器生成的序列，

密码分析不能预测序列的下一位和前一位，其安全性不是基于无人了解的复杂位发生器，而是基于因子分解 n 的数学理论，对于关心密码安全的伪随机位的人来说，这是最佳的选择。

这个算法的速度很慢，但有提高速度的方法，可以不只是使用每个 x_i 的最低位作为伪随机位，如果 x_i 的长度为 l，可以使用 x_i 的最低 $\log_2 l$ 位。

BBS 发生器十分慢不适用于流密码，然而，对于密钥分发之类的高安全性应用场合，这个发生器是最好的。

15.2.3 信息论方法

在流密码的这种方法中，假设密码分析者具有无限的时间和计算能力，抵抗这类敌手做到安全的唯一实用流密码是一次一密，由于在一个密码中各位的一次一密不实用，有时称之为一次一带(one-time tape)。二条磁带，加密端一条，解密端一条，在它们上面有着同样的随机密钥流。加密时简单地将明文与磁带上各位异或，解密时将密文与另一个相同的磁带上各位异或。由于密钥流中的各位是真正随机的，所以不会二次使用相同的密钥流中的各位。

U. Maurer 描述了一种基于把明文与多个大的公共随机位序列相异或的方法，在每个序列中密钥为起始位置，用一个根据攻击者具有多少存储器(而不是多少计算能力)的可计算的攻破概率，可以证明这个方法基本是安全的。Maurer 声称当具有100个不同的序列，每个序列有10^{20}个随机位时，这个方法是实用的。

Schnorr 设计的另一种流密码，假设密码分析只能访问有限的密文位，其结果是相当理论性的，至少目前还没有实用价值。

15.2.4 随机化流密码

在随机化流密码中，密码编制人员试图使密码分析不得不面临规模大到无法求解的问题。目标是增加密码分析需要处理的位数，同时保持密钥较小，这可以通过加密和解密使用大的公共随机串来实现，由密钥指定大的随机串中哪部分用于加密和解密，不知道密钥的密码分析者只能对整个随机串蛮力搜索。

这种密码的安全性可表示为密码分析者在确定密钥(而不是纯猜测)之前必须检查的平均位数。

尽管理论上对这类密码作了一些探讨，但还没有具体实现。J. Massey 和 I. Ingemarsson 提出了 Rip van Winkle 密码，之所以如此取名是由于接收者在试图解密之前需要接收2^n 位密文。如图15.15所示的这个算法是易于实现的、可证明是安全的但完全不实用，正如 Massey 所说："如果有人愿意等上数百万年来阅读明文，这个密码可保证密码分析需数千年时间才能破译。"

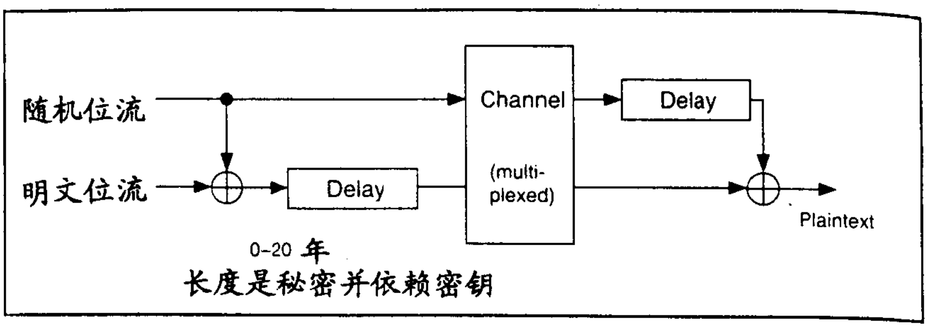


图15.15 Rip van Winkle 密码

U. Maurer 改进了这类方法，他的算法出自多个大的公共随机位序列的异或，密钥作为是每个序列中起始位置集合。可以证明这个方法基本是安全的：攻破算法的可能性是根据攻击者可用的存储器数量，而与计算能力无关。为使这个方法实用，需要约100个位序列，每一序列有10^{20}位。

15.3 真正随机序列发生器

伪随机数仅符合密码安全还是不够的，密码术中许多时候需要真正的随机数，密钥生成就是其中一例，基于 CSPRNG 生成随机密钥当然是好的，但如果敌手得到了 CSPRNG 的副本以及用于生成密钥的所有秘密信息，那么无论算法如何安全，敌手都能生成同样的密钥并攻破密码体制。随机序列发生器的序列是无法再生的，没有人能够再生这些发生器输出的位序列。

真正随机序列发生器是应该经过测试的，这方面有大量的文献，测试随机性的文献有，U. Maurer 证明了所有这些测试可由序列压缩而知，如果可以把随机序列压缩，则它不是真正随机的。

尽管对各种技术生成的随机序列不能证明，然而重要的是生成敌手不易猜测并不易于再生的位序列。

RAND 表

1955年，当时计算机还是很新奇，Rand 公司出版了一本含有100万个随机数的书，同时书中还描述了其生成方法：

“本书中的随机数是把由电子转盘生成的基本表再随机化而得，简单地说，一个平均每秒提供约100,000个脉冲的随机频率脉冲源由一个常量频率脉冲约每秒选通一次，脉冲标准化电路把脉冲传递过一个5位的二进制计数器。原则上这种机器是32级的转盘，平均每次试验约3000转，并且每秒生成一个数。用一个二进制-十进制转换器把32个数中的20个作转换（其余12个丢弃），并且只保留二位数中的最后一位，最后一位输入到 IBM 打孔机来最后生成随机数的打孔卡片表。”

此书接着讨论对这些数据各种随机性测试的结果，也建议如何用此书找到随机数：

“数字表行号为00000到19999。使用此表时首先要找到随机起始位置，一般是把书翻到数字表的任意一页并随意选择一个5位数，把该数的首位数模2，以此来确定起始行，原5位数的右二位模50来决定起始行中的起始列。为避免重复把书翻到同一页以及一个人往往在页中心选择数字，每个用于确定起始位置的5位数都给以标记，下次不能再用。”

这本书的主要内容是“随机数表”，表以5位为一组列出，如“10097 32533 76520 13586…”，一行50组并且每页50行，表共有400页，只有第283页有一个有趣的：“69696”。此外，读起来都很枯燥。书中还有一个100,000个正常偏差的表。

此书的有趣之处并不是有100万个随机数，而是这些数是在计算机革命之前生成的。许多密码算法采用任意的常量，称之为“魔数”(magic number)，从 RAND 表中选择魔数保证它没有被使用过，如 Khafre 所为。

使用计算机时钟

如果需要单个随机比特（或许多），可以取任何时钟寄存器的最低位，在 UNIX 系统里由于各种潜在的同步问题，这个方法也许不是非常随机，但在 PC 机上是比较理想的。

用这种方法取过多比特数据时要小心，连续多次执行同一程序往往会使这种方法生成的位数据变态，例如，如果每个位发生程序执行时取偶数个时钟，则发生器会输出无穷的相同位

数据流，如果每个位发生程序执行时取奇数个时钟，则发生器会输出无穷的交替位数据流。即使不是如此明显地共振，产生的位流也远非随机。

测量键盘等待时间

人们的打字模式是相当随机的，测量连续击键之间的时间间隔，并且取其最低位。由于在UNIX 终端上这些数据进入工作程序之前要经过过滤器及其它机制，因此不可行，但在 PC 机上可以。

理想情况下，每次击键只要收集一位随机数，收集更多时根据打字员的水平会产生不同的变异。然而这一技术有其局限，虽然可以让某人打100个词左右来生成一个密钥，但让他打100，000个词的短文来生成一次一密的密钥流是不可能的。

使用随机噪音

收集大量随机位的最好方法是选取真实世界中的自然随机性，这种方法往往需要专用硬件，但可以借助于计算机来实现。

找一个有规律但又随机的事件，如空气噪音的峰值，小孩蹒跚学步跌倒等等，测量并记录第一次事件与下一次事件之间的间隔，同样，测量并记录下一次事件与第三次事件之间的间隔，如果第一个时间间隔大于第二个时间间隔，则输出1作为一位，否则输出2作为事件。对下一位再进行同样处理。

G. B. Agnew 提出一种适合于集成到 VLSI 设备中的真正随机位发生器，它是一种金属绝缘半导体电容器(MISC)，把二个电容器靠近，随机位是二者电荷差的函数，另一个 LSI 随机数发生器基于振荡器内频率不稳定生成随机位流。AT&T 公司生产的一种商业芯片从同一现象中生成随机数，M. Gude 设计了一种从物理现象(如放射衰变)中收集随机位的随机数发生器，M. Richter 设计了一种基于半导体二极管中热噪音的随机数发生器。

另外还有使用计算机磁盘驱动器的随机数发生器，发生器测量读一个磁盘块所需的时间并把各种大小的时间用作随机数源，大部分磁盘旋转速度不同由空气涡流引起，因此存在一定的随机性。

在计算机系统中还有另外一些可用的随机源：

日期时间
实际 Mouse 位置
显示器当前扫描线数
实际显示图像的内容
FAT 和核心表等的内容
dev/tty 的访问/修改时间
CPU 加载
网络包的到达时间
麦克风的输入

问题不在于实际找到的随机源，而在于保持随机性的对随机源的采样方式。

偏移和相关

这些系统的主要问题是生成的序列内有相关和偏移，基本的物理过程可能是随机的，但是计算机和物理过程的数字部分之间有各类测量仪器，这些仪器可以很容易地引入偏移和相关，消除相关的一种方法是把几位异或在一起，如果随机位以 e 倍速率趋向于0，则0的概率可表示为：

$$P(0)=0.5+e$$

把二位异或在一起得到：

$$P(0)=(0.5+e)^2+(0.5+e)^2=0.5+2e^2$$

同样，把四位异或在一起得到：

$$P(0)=0.5+8e^4$$

把 m 位异或在一起将指数地收敛到一个0与1相当概率。

一种更好的方法可以从另外的随机源中去除所有的偏移，以成对的方式处理各位，如果各位相同，则丢弃并处理下一对，如果二位不同，取首位为发生器的输出。

二种方法都有一个潜在的问题，即如果相邻位之间有相关，则这些方法会“增加”偏移，改正这个问题的一种方法是使用多个随机源，取四个不同的随机源并把各位异或在一起，或者取二个随机源并以成对的方式处理各位。

例如，取放射源并在计算机中挂一个盖格计数器，取一对噪音二极管并把每次噪音超过某个值记录为一个事件，测量空气噪音，从每个事件中取一随机位并异或在一起生成随机位，其可能性是无限的。

随机数发生器有偏移并不一定意味着它是无用的，它仅仅意味着不安全。例如，考虑 A 生成三重 DES 的112比特密钥，她只有一个趋向0的随机位发生器：55％生成0和45％生成1，这意味着每位密钥只有0.99277位熵，如果发生器理想则熵为1。攻击者试图攻破这个密钥，优化他的蛮力搜索：首先试探最可能的密钥(000…0)，向最不可能的密钥(111…1)进行。由于偏移，可望在2^{109}次试探内找到密钥，如果没有偏移，则在2^{111}次试探内找到密钥，结果密钥不太安全，但不很严重。

用单向哈希函数扩散随机性

另一种技术是用单向哈希函数处理随机位，它具有从输入提取熵的效果，也有助于消除物理发生器的低级偏移或相关。

例如，如果为一个密钥需要128个随机位，使用上述任何资源生成大量的随机位并用一个好的单向哈希函数处理，随机位序列中的熵之和至少有128比特，则单向哈希函数的输出也应有128比特的熵，即理想随机，即使每位具有远小于1比特熵时也是成立的。

15.4 生成数与非均匀分布

如果对伪随机数或随机数要求位的分布比均匀分布更复杂，则从位着手进行工作。如果需要一个0到9之间的随机数，取4比特并忽略大于9的数。如果需要一个1到1,000,000之间的随机数，取20比特并忽略大于1,000,000的数(只有5％的机会出现)。

有一些简单得多的方法，就是用下列这些函数，假设已存在一个函数：

```
rand_bit()
```

它返回一个随机位。

生成一个无符号长整数：

```
unsigned long long_rand(void)
{
ulong v;
int i;
for (i=0; i<BITS.PER.ULONG;i++)
```

```
    {
    v<<=1;
    V|=rand_bit();
}
return v;
}
```

生成一个无符号短整数：

```
unsigned short short_rand(void)
{
return (long_rand()& 0xFFFF);
}
```

生成一个0.0到1.0之间的双精度数：

```
#define MAXLONG 4294967296.0
double frac_rand(void)
{
return (long_rand() / MAXLONG);
}
```

生成一个 a 到 b 之间的随机整数：

```
int range_rand(int a, int b)
{
return ((int)(long_rand() % (b-a+1) + a));
}
```

生成一个 double_a 到 double_b 之间的随机双精度数：

```
#define MAXLONG 4294967296.0
double range_double_rand(double double_a, double double_b)
{
return((long_rand()/MAXLONG) * (double_b - double_a)
        + double_a);
}
```

生成一个具有正态分布的随机双精度数，正态分布由给定均值和方差定义：

```
#define MAXLONG 4294967296.0
#define PI 3.141592653589793
double normal_rand(double mean, double std_dev)
{
double a,b,x;
a==long_rand()/MAXLONG;
b==long_rand()/MAXLONG;
x==sqrt(-2.0* log(a))*cos(2* PI*b);
return (mean + std_dev*x);
}
```

生成一个由给定均值定义的指数分布的随机双精度数：

```
#define MAXLONG 4294967296.0
double exponential_rand(double mean)
{
return (-mean * log(long_rand() / MAXLONG));
}
```

15.5 生成随机置换

给定一个随机数发生器，建立随机置换发生器是容易的，有时称它为洗牌，因为洗牌只是随机把牌置换，别无任何作用。

算法来自于 Durstenfeld，用 C 语言描述为：

```
void shuffle(int a[], int n)
    /* the array a[0…n-1] is of things to be shuffled */
{
int i,z,t;
for (i=n-2; i>=0; i--)
    {
        z=random(i);      /* This function produces a */
              /* random # between 0 and i */
        t = a[i];
        a[i] = a[z];
        a[z] = t;
    }
}
```

这个函数可以生成任何类型的随机置换，如果 random()函数是确定性的，则置换可以再生(甚至逆置)，如果 random()函数生成密码安全的随机序列，则产生置换也是密码安全的，如果 random()函数生成真正随机的序列，则置换也是真正随机的。

排序和洗牌在某种意义上是逆操作，如果有一个良好的排序算法，可以用来对一个文件做洗牌处理，例如把归并排序算法修改成凡要做判断的地方，用以随机位发生器的下一位而不是去检验是否 a<b，这就可把文件洗掉，如，如果文件有 n 项，算法用 nlogn 个随机位和 O(nlogn)的操作，这是相当有效的。

第16章　协议的特殊算法

16.1　密钥交换

Shamir 的三遍协议

由 Shamir 设计的这个协议使 A 和 B 安全地通讯，无须事先进行密钥和公钥的交换。

假设存在可交换的对称密码：

$$E_{KA}(E_{KB}(P))=E_{KB}(E_{KA}(P))$$

A 和 B 的私钥分别为 K_A 和 K_B，A 要把报文 P 送给 B，协议如下：

(1)A 用她的密钥加密明文 P 并把密文送给 B

$$C_1=E_{KA}(P)$$

(2)B 用他的密钥加密 C_1 并把密文送给 A

$$C_2=E_{KB}(E_{KA}(P))$$

(3)A 用她的密钥解密 C_2 并把密文送给 B

$$C_3=D_{KA}(E_{KB}(E_{KA}(P)))=D_{KA}(E_{KA}(E_{KB}(P)))=E_{KB}(P)$$

(4)B 用他的密钥解密 C_3 恢复明文报文 P。

一次一密是可交换的，安全性极好，但不能用于这个协议。对于一次一密，三个密文报文为：

$$C_1=P\oplus K_A$$

$$C_2=P\oplus K_A\oplus K_B$$

$$C_3=P\oplus K_B$$

窃听者在 A 和 B 之间传送报文时把三个报文记录下来，简单地将它们异或在一起即恢复了报文：

$$C_1\oplus C_2\oplus C_3=(P\oplus K_A)\oplus(P\oplus K_A\oplus K_B)\oplus(P\oplus K_B)=P$$

显然，这是不安全的。

Shimar 描述了一个类似于 RSA 的加密算法可以用来安全通信。设 p 为大素数且 p－1有一个大素数的因子(用于阻碍求解模 p 的离散对数)，像 RSA 一样，选择一个加密密钥 e 和解密密钥 d，加密密钥 e 要与 p－1互素。如果现有一个素数测试程序而不是最大公因子程序，最简单的方法是为 e 选择一个大素数，用欧几里德算法求解满足 $de=1(\bmod p-1)$ 的 d。

加密报文时计算：

$$C=P^e(\bmod p)$$

解密报文时计算：

$$P=C^d(\bmod p)$$

不求解离散对数问题，窃听者似乎无法恢复 P，但类似于公钥密码术中许多问题一样，这还没有得到证明。

这个协议完成了一些简单的公钥加密没有完成的功能，它允许 A 初始化与 B 的安全通讯，而无需知道他的密钥。对于 A 使用公钥算法，她要知道 B 的公钥，用 Shamir 的三遍协议，

她只要把密文报文送给 B。同样的事用公钥算法执行则有以下的步骤：

(1)A 向 B(或 KDC)要他的公钥

(2)B(或 KDC)把公钥送给 A

(3)A 用 B 的公钥加密 P 并送给 B

COMSET

COMSET(COMmunications SETup)是为 RIPE 项目设计的共同的标识和密钥更换协议，采用公钥密码术，COMSET 允许 A 和 B 他们可相互确认对方，也可以更换私钥。

COMSET 的数学原理为 Rabin 算法。

16.2 加密密钥更换

加密密钥更换(EKE)协议由 S. Bellovin 和 M. Merritt 设计，它提供计算机网络上的安全性及认证能力，以一种新奇的方式同时采用对称和公钥密码术：共享的私钥用于加密随机生成的公钥。

基本协议

A 和 B 共用一个口令 P，采用这个协议，他们可以相互认证并生成公用的会话密钥 K。

(1)A 生成一个随机公钥 K′，并用对称算法及密钥 P 加密：$E_P(K')$，送给 B：

$A, E_P(K')$

(2)B 知道 P，把报文解密得到 K′，再生成一个随机会话密钥 K，并用从 A 接收到的公钥和私钥 P 加密：

$E_P(E_{K'}(K))$

并送给 A。

(3)A 把报文解密得到 K，并生成一个随机串 R_A，用 K 加密，把结果送给 B：

$E_K(R_A)$

(4)B 把报文解密得到 R_A，并生成另一个随机串 R_B，用 K 加密二个随机串，把结果送给 A：

$E_K(R_A, R_B)$

(5)A 把报文解密得到 R_A 和 R_B。假如从 B 接收到的 R_A 与第(3)步中送给 B 的串相同，则用 K 加密 R_B 并送给 B：

$E_K(R_B)$

(6)B 把报文解密得到 R_B。假如从 A 接收到的 R_B 与第(4)步中送给 A 的串相同，协议结束，双方把 K 作为会话密钥通讯。

第(3)步中，A 和 B 都知道 K′和 K，K 为会话密钥，可用于加密二者之间的所有报文。在 A 和 B 之间的其他人(如窃听者)只知道 $E_P(K')$，$E_P(E_{K'}(K))$以及用 K 加密的一些报文。在其它协议中，窃听者可对 P 作猜测(人们往往选择不好的口令，如果窃听者聪明的话，她可做出良好的猜测)，并测试其猜测。在本协议中，窃听者在没有攻破公钥算法时无法测试其猜测，并且如果 K′和 K 都是随机选择的，这是个无法超越的问题。

协议中的“询问-应答”部分，即第(3)到第(6)步，使协议生效。第(3)到第(5)步向 A 证明 B 知道 K，第(4)到第(6)步向 B 证明 A 知道 K。Kerberos 协议的时戳交换也是完成同样的步骤。

用 RSA 算法实现 EKE

对用 RSA 算法实现 EKE 来说，RSA 算法似乎是完美的，但也有一些小问题。作者推荐在

第(1)步中只对加密指数加密并显式传送模数。

用 ElGamal 算法实现 EKE

用 ElGamal 算法实现 EKE 是很显然的,并可简化基本协议。记 g 和 p 为部分公钥并为所有用户公用.私钥为随机数 r,公钥为 g_r(mod p)。第(1)步中 A 送给 B 的报文为:

$$A, g^r \pmod p$$

注意这个公钥不要用 P 加密,一般来说这是不可以的,但对于 ElGamal 算法是可以的。

B 选择一个随机数 R(这是专为 ElGamal 算法而做,独立于为 EKE 选择的所有随机数),第(2)步中送给 A 的报文为:

$$P(g^R \pmod p), Kg^{Rr} \pmod p))$$

用 Diffie-Hellman 协议实现 EKE

用 Diffie-Hellman 协议时,K 是自动生成的,最后的协议更为简单。为网络上所有用户而设置了为 g 和 n 的一个值。

(1)A 选取一个随机数 r_A,送给 B:

$$A, g_A{}^r \pmod n$$

用 Diffie-Hellman 协议时,A 不要用 P 加密她的第一个报文。

(2)B 选取一个随机数 r_B,并计算:

$$K = g_A{}^r * r_B \pmod n$$

他生成一个随机串 r_B,再计算下式并送给 A:

$$P(g_{B^r} \pmod p)), K(R_B)$$

(3)A 把 B 的报文的前一半解密得到 g_{B^r}(mod p),再计算 K 并用 K 解密 r_B。她生成另一个随机串 R_A,用 K 加密二个随机串,并把结果送给 B:

$$K(R_A, R_B)$$

(4)B 把报文解密得到 R_A 和 R_B,假如从 A 接收到的 R_B 与在第(2)步中送给 A 的串相同,则用 K 加密 R_A 并送给 A:

$$K(R_B)$$

(5)B 把报文解密得到 R_A,假如从 A 接收到的 R_A 与在第(3)步中送给 B 的串相同,协议结束,双方用 K 作为会话密钥通讯。

增强型 EKE

Bellovin 和 Merritt 建议增强协议中"询问-应答"部分,以防止如果密码分析者恢复了老的 K 值而引起的可能攻击。

在基本 EKE 协议的第(3)步中,A 生成另一个随机数 S_A,送给 B:

$$K(R_A, S_A)$$

第(4)步中,B 生成另一个随机数 S_B,送给 A:

$$K(R_A, R_B, S_A)$$

于是 A 和 B 都可以计算真正的会话密钥 S_A XOR S_A,这个密钥用于 A 和 B 之间以后所有的报文,K 只用作密钥交换的密钥。

由于 P 从未用于加密会直接导致 S 的任何内容,S 值暴露不会给窃听者窃听者任何有关 P 的信息。对 K 密码攻击也是不可行的:K 只用于加密随机数据,并且 S 也从未直接加密。

EKE 的应用

Bellovin 和 Merritt 建议这个协议可用于安全的公共电话,另外,EKE 的主要优点是可以

同时使用对称和公钥密码术，并由此增强了双方的密码功能。

16.3 多钥、公钥密码术

这个算法是 RSA 的推广。模数 n 为二个素数 p 和 q 的乘积，然而，不是选取满足 $ed=1 \bmod (p-1)*(q-1)$的 e 和 d，而是选取 t 个密钥 k_i，使得：

$$k_1 * k_2 * \cdots * k_t = 1 \bmod (p-1)*(q-1)$$

由于

$$M^{k1*k2*\cdots*kt}=M$$

这是一个多钥算法。

例如，现有5个密钥，以 k_3和 k_5加密的报文可用 k_1，k_2和 k_4解密：

$$C=M^{k3*k5} \bmod n$$

$$M=C^{k1*k2*k4} \bmod n$$

16.4 秘密广播

A 要从某发送器广播报文 M，但是，她不想让每位听众都听懂，事实上，她只想让一部分人能够恢复密文 S。

A 可与每位听众共享一个不同的密钥(私钥或公钥)，她用随机密钥 K 加密报文，再用所有有关的接收者的密钥加密 K，每次形成一个经加密的 K 的不同副本，最后，她广播加了密的报文及所有加了密的 K。某位听众 B，或者试图用他的私钥将所有 K 解密，(如果 A 不在意听众都知道她的报文是给谁的)或者寻找跟随在加了密的密钥后的他的姓名。多钥密码术以这样方式工作。

另一种方法是首先每位听众与 A 共享一个比所有加了密的报文都大的私钥，她用随机密钥 K 加密报文。接着，她计算一个整数 R，使得当某私钥能解密报文时，R 与该私钥求模同余为 K，否则同余为0。

例如，如果 A 希望 B，C 和 E 收到秘密，而 D 和 F 不能收到，她用 K 加密报文并这样计算 R：

$$R \bmod K_B=K$$

$$R \bmod K_C=K$$

$$R \bmod K_D=0$$

$$R \bmod K_E=K$$

$$R \bmod K_F=0$$

这是个简单的代数问题，A 可轻易求解。当听众收到广播时，计算接受到的密钥与他们的私钥求模，如果他们是能接受报文的听众，则可恢复密钥(K)，否则一无所获。

由 S. Berkovitz 设计的第三种方法采用阈值方法，像其它方法一样，每位潜在的接收者得到一个私钥，这个密钥是在尚未建立的阈值机制中的影条(shadow)。为了在系统中增加一些随机性，A 为她自己保存了一些私钥。假设共有 K 个听众。

接着广播 M，A 用密钥 K 加密 M 并执行下列步骤：

(1)A 选择一个随机数 j，这个数用于隐藏报文接收者的数量，无需太大，甚至可以小到0。

(2)A 用：

K 作为密钥

有意接收者的私钥作为影条

非接收者的私钥不作为影条

j 个与所有私钥都不同的随机选择影条

建立(k+j+1,2k+j+1)阈值机制。

(3)A 广播 k+j 个随机选择的影条,其中没有一个是第(2)步中列出的影条。

(4)接收到广播的所有听众把他们的影条加到他们接收到的 k+j 个影条中,如果影条相加允许他们计算秘密,则他们已恢复了密钥,否则还没有。

16.5 秘密共享算法

秘密共享算法也称为阈值(threshold)机制,工作过程如下:报文分成 n 片,称为影条(shadow),并使 m 个影条可重构报文,但不能比 m 少1,即任何 m-1个影条不能重构报文,这称为(m,n)阈值方法。

虽然这里叙述四种不同的方法,但 Kothari 证明了它们不过是一个更为一般的特例。

拉格朗日(LaGrange)多项式机制

A. Shamir 用有限域内多项式方程构造了一种阈值机制,首先选择一个素数 p,要求大于可能的影条数和私钥最大长度,为共享秘密,生成一个 m-1阶任意多项式。例如,如果要建立(3,n)阈值机制(重构 M 需要3个影条),生成一个平方多项式:

$$ax^2+bx+M(\text{mod } p)$$

其中 p 为大于所有系数的随机素数,系数 a 和 b 随机选取,M 为报文,素数必须公开。

在 n 个不同点估计多项式而得影条:

$$k_i=F(x_i)$$

即第一个影条为 x=1点估计的多项式,第二个影条为 x=2点估计的多项式等等。

由于平方多项式有3个未知系数 a,b 和 m,任何三个影条都可用于建立三个方程,由线性代数可解得 M,而二个影条不行,一个影条也不行,四个或五个影条是多余的。

例如,设 M 为11,为构造(3,5)阈值机制,即5位人中任何3位都可重构 M,首先生成一个二次方方程(7和8是随机选取的):

$$F(x)=7x^2+8x+11(\text{mod } 13)$$

5个影条为:

$$k1=F(1)=7+8+11=0(\text{mod } 13)$$
$$k2=F(2)=28+16+11=3(\text{mod } 13)$$
$$k3=F(3)=63+24+11=7(\text{mod } 13)$$
$$k4=F(4)=112+32+11=12(\text{mod } 13)$$
$$k5=F(5)=175+40+11=5(\text{mod } 13)$$

为了从3个影条中重构 M,例如取 k_2,k_3和 k_5,求解下列线性方程:

$$3=a*2^2+b*2+M(\text{mod } 13)$$
$$7=a*2^3+b*3+M(\text{mod } 13)$$
$$5=a*2^5+b*5+M(\text{mod } 13)$$

解答为 a=7,b=8和 M=11,故恢复了 M。

这个共享算法可方便地用大数实现。如果要把报文分成30等份,并且任何6份可重构 M,30位人中的每一位给以5次多项式求值:

$$F(x)=ax^6+bx^5+cx^4+dx^3+ex^2+fx+M(\bmod p)$$

6人有6个5次多项式可求解6个未知数(包括 M),5人则不能求解出 M。

这个秘密共享算法中最惊奇之处是如果各系数是随机选取的,具有无限计算能力的5人只能得到报文的长度(而且是人人都知道的),这与一次一密同样安全,穷举搜索(试探所有可能的第6个影条)会得出任何想及的报文都可能是秘密,这也适用4人秘密共享算法。

矢量机制

Blakley 提出的方法采用了空间中点的概念,把报文定义为 m 维空间中的一点,每个影条为包含该点的 m－1维超平面方程,任何 m 个超平面的交叉正好是该点。

例如,如果重构报文需要三个影条,则这个报文是三维空间中的一点,每个影子为不同的面。有了一个影条,可以知道这个点是在这个面上,有了二个影条,可以知道这个点是在这二个面的交线上,有了三个影条,就可以精确地确定点的位置:三个面的交点。

Asmuth-Bloom 机制

Asmuth 和 Bloom 用素数设计了另一种方法,对于(m,n)阈值机制,选择一个大于 M 的素数 p,再选择小于 p 的 n 个数 $d_1,d_2\cdots d_n$,使得各个 d 值递增有序,即 $d_i<d_{i+1}$,并且所有的 d 值都两两互素。

$$d_1*d_2*\cdots*d_m>p*d_{n-m+2}*d_{n-m+3}*d_n$$

分布各影条时,首先选择一个随机数 r,并计算:

$$M'=M+rp$$

影条 k_i 为:

$$k_i=M'(\bmod di)$$

采用中国余数定理,任何 m 个影条可以重构 M,但任何 m－1个影条则不成。

Karmin-Greene-Hellman 机制

Karmin,Greene 和 Hellman 设计的机制采用矩阵相乘。选择 n+1个 m 维矢量 $V_0,V_1\cdots V_n$,使得任何可能的 $m\times n$ 矩阵都构成秩为 m 的矩阵,矢量 U 为 m－1维的行向量。

M 为矩阵乘积 UV_0,各个影条为乘积 UV_i,其中 $i=1,2,\cdots,n$。

任何 m 个影条都可用于求解 $m\times m$ 线性方程系统,其中未知数为 U 的系数,就可计算出 UV_0,任何 m－1个影条不能求解这个线性方程系统,故不能恢复秘密。

高级阈值机制

前面各例只是说明了最简单的阈值机制:把秘密分成 n 个影条,使得任何 m 个影子都可用于恢复秘密。下面例子中采用 Shamir 的算法,当然前面各个算法也都是可用的。

如果要建立一种某人比其他人更重要的算法,则给那人更多的影条。设重构秘密要5个影条,如果某人有3个不同的影条,而其他人只有1个,则那人和其他2人即可重构秘密。如果不用那人,需要5人才能重构。

也可以有2人或更多人有多个影条,每个人可以有不同数量的影条。无论影条如何分配,任何 m 个影条可用于重构秘密。如果某人有 m－1个影条,无论是一个人或多少人都不能重构秘密。

也可以建立其它类型的机制。想象有二个敌对组可以共享一个秘密,重构这个秘密需要 A 组7人中的2人和 B 组12人中的3人,构造一个由线性方程和二次方程之积的3次多项式,给 A 组中每人一个线性方程值的影条,给 B 组中每人一个二次方程值的影条。

A 组中任何二个影条可用于重构线性方程,但该组中无论有多少其它影条都不能得到任

何有关秘密的信息，对于 B 组也一样：可用三个影条重构二次方程，但不能得到重构秘密所需的其它信息，只有当二组共享方程时，它们相乘才能重构秘密。

一般而言，任何可以想象的共享算法都可实现，只要设计与算法相关的方程系统即可。

与欺骗者共享秘密

这个算法修改了标准(m,n)阈值算法，以检测其中的欺骗者，它可采用任何算法，但这里以 LaGrange 算法说明。

选择一个大于 n 且大于

$$(s-1)*(m-1)/e+m$$

的素数 p，其中 s 是最大的可能秘密，e 为欺骗成功的可能性，可以使 e 任意小，它只是使数学更为复杂一些。像前面一样构造影条，只是 xi 不采用1,2,3,…,n，而采用1到 p－1之间的随机数。

于是，当攻击者带着一份假影条偷偷进入秘密重构过程时，他的不可能的概率极高，这就暴露了它是假的。

然而，当骗子揭穿时，他仍然知道了秘密(假设有 m 个其他有效的份)，另一个协议防止了这种情况，其基本思想是有一系列 k 个秘密，没有参与者事先知道哪个是正确的秘密，除真正的秘密之外，每个秘密都比前者大，参与者把各影条组合起来逐个生成秘密，直到生成比前一秘密小的秘密，这就是正确的秘密。

这个算法在生成秘密之前就可揭穿骗子，当参与者每次传送一个影条时会有些复杂。

16.6 难以觉察的信道

Ong-Schnorr-Shamir 算法

由 G. Simmons 设计的这个难以觉察的(subliminal)信道算法采用 Ong-Schnorr-Shamir 身份认证算法，类似于原算法，发送者(A)选择一个公开模数 n 和私钥 k，使得 n 和 k 互素，与原算法不同的是 k 由 A 和 B 共享(B 为难以觉察报文的接收者)。

公钥以同样的方法计算：

$$h=-k^{-2}(\bmod n)$$

如果 A 要通过无关紧要报文 M′传送难以觉察的报文 M，她首先确证 M′与 n 互素，并且 M 与 n 互素。

A 计算：

$$S_1=1/2*(M'/M+M)(\bmod n)$$

$$S_2=k/2*(M'/M-M)(\bmod n)$$

S_1和 S_2对一起为传统 Ong-Schnorr-Shamir 算法下的签名和难以觉察报文的载体。

认证报文时验证下式是否成立：

$$M'=S_1{}^2-S_2{}^2/k^2(\bmod n)$$

如果报文是真实的，接收者能用下列公式恢复难以觉察的报文：

$M=M'/(S_1+S_2/k)(\bmod n)$这个做法虽然可行，但是请记住 Ong-Schnorr-Shamir 算法已被攻破。

ElGamal 算法

Simmons 的第二个难以觉察的信道基于 ElGamal 签名算法。

密钥生成与基本的 ElGamal 签名算法相同，首先选择一个素数 p 以及二个都小于 p 的随

机数 g 和 r，再计算：

$$K=g^r \pmod p$$

公钥为 K，g 和 p，私钥为 r，除 A 之外，B 也知道 r，这是用于发送、阅读难以觉察报文的密钥，也是用于签名无关紧要报文的密钥。

发送难以觉察报文 M 时用无关紧要报文 M′，M 与 p 必须互素，M 与 p－1也必须互素，A 计算：

$$X=g^M \pmod p$$

并用欧几里德算法求解下列方程中的 Y：

$$M'=rX+MY \pmod p$$

与基本 ElGamal 算法一样，签名为 X 和 Y 对。

Walter 能验证 ElGamal 签名，他验证下式是否成立：

$$K^X X^Y \pmod p = g^{M'} \pmod p$$

B 能恢复难以觉察的报文，首先他验证：

$$(g^r)^X X^Y \pmod p = g^{M'} \pmod p$$

如果成立，他接受这个报文是真的（不是来自 W 的）。

接着恢复 M 时计算：

$$M=Y^{-1}(M'-rX) \pmod{p-1}$$

例如，设 p＝11，g＝2，私钥 r 选取8，即 Walter 可用于验证签名的公钥为 $g^r \pmod p = 2^8 \pmod{11} = 3$。

发送难以觉察报文 M＝9时，用无关紧要报文 M′＝5，A 验证9和11互素，5和11互素，9和11－1也互素，于是计算：

$$X=g^M \pmod p = 2^9 \pmod{11} = 6$$

求解下面方程中的 Y：

$$5=8*6+9*Y \pmod{10}$$

Y＝3，于是签名为 X 和 Y 对：6和3。

B 验证：

$$(2^8)^6 6^3 \pmod{11} = 2^5 \pmod{11}$$

成立，于是他恢复了难以觉察的报文：

$$\begin{aligned} M &= Y^{-1}(M'-rX) \pmod{p-1} \\ &= 3^{-1}(5-8*6) \pmod{10} \\ &= 7(7) \pmod{10} = 49 \pmod{10} = 9 \end{aligned}$$

ESIGN 算法

难以觉察的信道可添加到 ESIGN 算法中。

ESIGN 算法中，私钥为一对大素数 p 和 q，公钥为 $n=p^2q$，有了难以觉察的信道之后，私钥为三个素数：p，q 和 r，公钥为 $n=p^2qr$，变量 r 为 B 阅读难以觉察报文时所需的另外一部分信息。

签名一般的报文时，A 首先选取一个随机数 x＜pqr，并计算：

w：大于 $(H(m)-x^k \bmod n)/pqr$ 的最小整数

$$s=x+((w/kx^{k-1}) \bmod p)pqr$$

H(m) 为报文的哈希值，k 为安全系数，s 值为签名。

验证签名时，B 计算 $s^k \bmod n$，也计算大于 n 除以3的比特数的最小整数，如果 H(m) 小于等于 $s^k \bmod n$，并且 $s^k \bmod n$ 小于 $H(m)+2^a$，则认为签名有效。

发送难以觉察报文 M 时用无关紧要报文 M′，A 用 M 而不是 H(m) 计算 s，表示报文必须小于 p^2qr，她再选择一个随机数 u 并计算：

$$x' = M' + ur$$

再使用 x′ 作为随机数 x 来签名 M′，第二个 s 值作为签名而发送。

W 能够验证 s(第二个)为 M′的有效签名。

B 也能同样认证报文，但由于他知道 r，他可计算：

$$s = x' + ypqr = M' + ur + ypqr = M \pmod r$$

这个难以觉察信道的实现比前二个好得多，在 Ong-Schnorr-Shamir 和 ElGamal 实现中，B 有 A 的私钥，除了能阅读来自 A 的难以觉察的报文之外，B 能冒充 A 并签名一般的文档，A 对此毫无办法。

ESIGN 算法没有这个问题，A 的私钥为三个素数 p，q 和 r，B 的私钥为 r，他知道 $n=p^2qr$，但是要恢复 p 和 q，他得将 n 因子分解，如果各素数足够大，B 冒充 A 同样会有许多麻烦。

DSA

DSA 中也有几个难以觉察的信道，最简单的难以觉察的信道涉及选择 k，这应该是160比特的随机数，然而，如果 A 选择了特殊的 k，则知道 A 私钥的 B 能找到 k，A 在每个 DSA 签名中可送给 B 一个160比特的难以觉察的报文，其他每个人都可简便地验证 A 的签名。由于 k 为随机数，A 和 B 需要共用一次一密算法加密难以觉察报文以生成 k，并且象 ElGamal 难以觉察信道算法，B 必须知道 A 的私钥。

DSA 中也有不需 B 知道 A 私钥的难以觉察的信道，它们也涉及选择特殊的 k 值，但不能用于发送160比特的信息，这里的算法允许 A 和 B 每个签了名的报文交换1比特难以觉察的信息：

(1)A 和 B 取一个共同的随机素数 P(与签名算法中的参数 p 不同)，这是难以觉察信道的私钥。

(2)A 签名无关紧要报文 M，如果 A 要送给 B 难以觉察的位1，她要确保签名的 r 参数是模 P 的平方余数，如果要送给 B 一个0，她要确保签名的 r 参数是模 P 的平方非余数。A 重复用随机的 k 值签名报文，直到得到具有所需特征 r 的签名。由于平方余数和平方非余数的可能性相同，这也并不太难。

(3)A 把签了名的报文送给 B

(4)B 验证签名以保证报文是真实的，再验证 r 是模 P 的平方余数还是平方非余数以及恢复难以觉察的位。

通过将 r 成为模各种参数的平方余数或平方非余数，可以传送多位。

这个算法可扩展为每个签名传送多个难以觉察的位，如果 A 和 B 同取二个随机素数 P 和 Q，A 通过选择一个随机数 k，使得 r 为模 P 的平方余数或平方非余数，或者是模 Q 的平方余数或平方非余数，可以传送二位。随机值 k 有25%的机会生成正确形式的 r。

下面是窃听者，一位卑劣的 DSS 实现者，如何在每次签名文档时使算法发送10位 A 的私钥：

(1)窃听者将 DSS 封装在 VLSI 芯片中，这样没人能检查其内部工作情况。他在制成的 DES 中建立14位的难以觉察的信道，即选择14个随机素数，并让芯片选择一个 k 值，使得14个

素数中的每一个求模后，或者是 g 次方余数，或者是平方余数(取决于难以觉察的报文)。

(2)窃听者把芯片分给 A，B 以其他人。

(3)A 用她的160位私钥 y 签名报文。

(4)芯片随机选择一个 y 的10位块：如第一个10位，第二个10位等等。由于有16个10位块，用4位可以标识哪一块，这4位标识符加上10位密钥构成14位难以觉察的报文。

(5)芯片测试随机值 k，直到找到具有传送难以觉察报文正确的平方余数特征的 k 值，一个随机的 k 值是正确者的可能性为1/16，384，假设芯片每秒能测试10，000个 k 值，找到一个正确的 k 需要2秒少一点，这个计算还不涉及报文以及可在 A 签名报文之前离线执行。

(6)芯片用第(5)步选择的 k 值签名报文。

(7)A 把数字签名送给 B，或者在网络上公开。

(8)窃听者恢复 r 值，并且由于他知道14个素数，可以把难以觉察的报文解密。

这个算法只要14个素数没有泄露，窃听者就一直是安全的。

其它算法

还有其它把难以觉察的信道嵌入签名协议中的算法，例如把难以觉察的信道嵌入 Fiat-Shamir 协议和 Feige-Fiat-Shamir 协议，事实上，所有的签名协议都可转换为难以觉察的信道。

16.7 不可否认的数字签名

这个算法由 D. Chaum 设计，首先公开一个大素数 p 和一个本原元 g，由一组签名者使用。A 有一个私钥 x 和一个公钥 g^x。

签名报文时，A 只需计算 $z=m^x$。

验证过程稍微复杂一些：

(1)B 选择二个都小于 p 的随机数 a 和 b，并把

$$c=z^a(g^x)^b(\text{mod } p)$$

送给 A。

(2)A 计算 $x^{-1}(\text{mod } p-1)$，并把

$$d=c^{x^{-1}}(\text{mod } p)$$

送给 B。

(3)B 验证下式是否成立：

$$d=m^{\varepsilon}g^b(\text{mod } p)$$

如果成立，则认为签名是真实的。

假如 A 和 B 执行这个协议，并且 B 相信 A 在报文上签了名，B 要使 C 相信，就把协议的副本给 C 看，D 要 C 相信一个由其他人签名的文件，他建立一个假的协议副本，首先生成第(1)步中的报文，再执行第(3)步中的计算生成 d 和用来自其他人的假传送作为协议的第(2)步，最后建立第(3)步中的报文。在 C 看来，B 和 D 的副本是相同的，除非她自己执行整个协议，她不能认清签名的有效性。

当然，如果当 B 完成协议时 C 一直盯着，其执行协议的实际次序，她就可以明白真假。

另一个协议由 Chaum 提出，这个协议不仅有一个确认协议，A 通过它使 B 相信有效的签名，还有一个否认协议，其中 A 用零知识交互协议让 B 判明作假的签名。

与前面的协议一样，公开大素数 p 和本原元 g，由一组签名者使用，A 有私钥 x 和公钥 g^x，

签名报文时，A 计算 $z=m^x$。

验证签名的过程如下：

(1)B 选择二个都小于 p 的随机数 a 和 b，并把

$$c=m^a g^b$$

送给 A。

(2)A 选择一个小于 p 的随机数 q，并把

$$s_1=c*g^q \quad s_2=(c*g^q)^x$$

送给 B。

(3)B 把 a 和 b 送给 A，这样 A 可确认 B 在第(1)步中没有欺骗。

(4)A 把 q 送给 B，这样 B 能使用 m^x 以及重构 s_1 和 s_2，如果

$$s_1=c*g^q$$

$$s_2=(g^x)^{b+q*}z^a$$

成立，则签名有效。

A 可能否认报文 m 的签名 z。

可转换的不可否认签名

这是可转换的不可否认签名的算法，这种签名可以验证、否认，也可转换为常规数字签名，它是基于 ElGamal 数字签名算法。

类似于 ElGamal 算法，首先选择二个素数 p 和 q，使得 q 能除尽 p－1。再要生成一个小于 q 的数 g，首先选择一个在2到 p－1之间的随机数 h，计算：

$$g=h(p-1)/q$$

如果 g 等于身份标识，则选择另一个随机数 h。

私钥为二个都小于 q 的不同随机数 x 和 z，公钥为 p，q，g，y 和 u，其中：

$$y=g^x(\mathrm{mod}\ p)$$

$$u=g^z(\mathrm{mod}\ p)$$

计算报文 m(其实为报文的哈希值)的可转换的不可否认签名时，首先选择一个在1到 q－1之间的随机数 t，再计算：

$$T=g^t \bmod p$$

$$m'=Ttzm \bmod q$$

于是计算 m′上的标准 ElGamal 签名。选择一个随机数 R，使得 R 小于 p－1并与 p－1互素，再计算 $r=g^R$，并用欧几里德算法计算下式的 s：

$$m'=rx+Rs(\mathrm{mod}\ q)$$

最后选择另一个随机数 t，签名为 ElGamal 签名(r，s)和 T。

以下是 A 如何向 B 验证她的签名：

(1)B 生成二个随机数 a 和 b，他计算：

$$c=T^{Tma}g^b(\mathrm{mod}\ p)$$

并送给 A。

(2)A 生成随机数 k，并计算：

$$h_1=cg^k(\mathrm{mod}\ p)$$

$$h_2=h_1{}^z(\mathrm{mod}\ p)$$

并送给 B。

(3)B 把 a 和 b 送给 A。

(4)A 验证：

$$c = T^{Tma} g^{b} \pmod p$$

并把 k 送给 B。

(5)B 验证：

$$h_1 = T^{Tma} g^{b+k} \pmod p$$

$$H_2 = y^{ra} r^{sa} u^{b+k} \pmod p$$

A 可公开 z 而把所有不可否认的签名转换为一般的签名，于是任何人都可独自验证她的签名。

T. Pederson 把这个思想和秘密共享算法组合建立了分布式的可转换不可否认的签名，人们可签名报文并分布确认签名有效的能力，例如，他们可能需要5人中的3人参与协议，以让 B 相信签名有效。

16.8 加密数据的计算处理

离散对数问题

现有一个大素数 p 和一个生成元 g，A 有一个特殊的 x 值，并要知道 e，使得

$$g^{e} = x \bmod p$$

这是一般的难问题，假设 A 没有求解所需的计算能力，而 B 有能力求解(例如 B 在政府或大的计算机构里)，下面是 A 如何让 B 求解而不暴露 X：

(1)A 选择一个小于 p 的随机数 r。

(2)A 计算：

$$x' = xg^{r} \bmod p$$

(3)A 请求 B 求解：

$$g^{e'} = x' \bmod p$$

(4)B 求解 e′并送给 A。

(5)A 通过计算：

$$e = e' - r \bmod p-1$$

而求解 e。

还有对于平方余数问题和本原根问题的类似协议。

16.9 公正掷币

下面各协议允许 A 和 B 可通过 Modem 通信而进行公正掷币，这是向井中掷币的例子。首先，只有 B 知道掷币结果并告诉给 A，后来 A 可以检验 B 告诉她的掷币结果是否正确。

用平方根掷币

掷币子协议如下：

(1)A 选取二个大素数 p 和 q，并计算它们的积：

$$n = pq$$

把 n 送给 B。

(2)B 选取一个小于等于 n/2的随机正整数 r，并计算：

$$z = r^{2} \bmod n$$

把 z 送给 A。

(3)由于 A 知道 n 的因子分解,她可计算 z(mod n)的4个平方根,称之为+x,-x,+y 和-y。令 x' 为+x mod n 和-x mod n 的小者,y' 为+y mod n 和-y mod n 的小者,注意 r 要么等于 x',要么等于 y'。

(4) A 猜测 $r=x'$ 还是 $r=y'$,她把1位猜测信息送给 B(她的做法是找到 x' 和 y' 不同的最小位,再告诉 B"r 的第 i 位为0"或者"r 的第 i 位为1")。

(5) 如果 A 猜中,则掷币结果为正面,否则为反面。B 宣布结果。

验证子协议如下:

(6)B 把 r 送给 A。

(7)A 把 p 和 q 送给 B。

由于 A 无法知道 r,因此她确实是猜测,她在第(4)步中只告诉 B 一位猜测信息,以防止让 B 知道 x' 和 y'。如果 B 有了 x' 和 y',他就可因子分解 n,因而在第(4)步之后改变 u,这个协议的安全性在于因子分解是很困难的。

用模 p 的指数掷币

在这个协议中,指数对素数 P 求模用作为单向哈希函数。

掷币子协议如下:

(1)A 和 B 共同选择一个素数 p,使得 p-1的因子分解已知。

(2)B 在 GF(p)内选取二个本原元 h 和 t,并送给 A。

(3)A 选择一个与 p-1互素的随机整数 x,她计算下面二个式子之一:

$$y=h^x \pmod p$$

$$y=t^x \pmod p$$

把 y 送给 B。

(4)B 猜测 y 是 h 的函数还是 t 的函数,把猜测结果送给 A。

(5)如果 B 猜中,掷币结果为正面,否则为反面。A 宣布掷币结果。

验证子协议如下:

(6)A 把 x 告诉给 B,B 计算 $h^x \pmod p$ 和 $t^x \pmod p$,既确认 A 的公正性又验证掷币结果。

第(1)步可以这样完成:A 和 B 各建议一些大素数,并把它们乘在一起,再验证乘积乘以2加1是否为素数。已知 p-1的因子分解有助于计算模 p 的离散对数,B 要求 p-1不是由小的素数因子构成。

如果 A 要行骗的话,她得知道二个整数 x 和 x',使得 $h^x=t^{x'} \pmod p$。给定 x,为了计算 x',她得计算:

$$\log_t h^x \pmod p$$

如果 A 知道 log th,她也许会这么做,但是 B 在第(2)步中选取 h 和 t,A 除了试图计算离散对数之外,别无其它方法。如果 A 选择了与 p-1不互素的 x 值,B 在第(6)步中也可检测出来。

如果 h 和 t 不是 GF(p)中的本原元,B 可以欺骗 A,但是由于 A 知道 p-1的素数因子分解,她可在第(2)步之后方便地检测出来。

这个协议的优点之一是如果 A 和 B 要多次掷币,他们可采用同样的 p,h 和 t 值,A 只要生成一个新的 x,并且从第(3)步开始继续执行协议。

用 Blum 整数掷币

Blum 整数可用于掷币协议中：

(1)A 生成一个 Blum 整数 n 和一个与 n 互素的随机数 x，$x_0=x^2 \bmod n$，$x_1=x_0^2 \bmod n$，把 n 和 x_1送给 B。

(2)B 猜测 x_0是奇数还是偶数。

(3)A 把 x 和 x_0送给 B。

(4)B 检验 n 为 Blum 整数(A 要把 n 的因子给 B，或者执行零知识协议以让 B 相信 n 为 Blum 整数)，并且验证 $x_0=x^2 \bmod n$，$x_1=x_0^2 \bmod n$，如果所有验证都成立，如果他猜中，则 B 胜利。

n 为 Blum 整数是很容易的，否则，A 能找到满足 $x_0=x^2 \bmod n=x'^2 \bmod n$ 的 x'，其中 x_0^2 也是平方余数。如果 x 为偶数而 x'为奇数(反之亦然)，A 完全可以欺骗 B。

16.10　公正的密码系统

公正的 Diffie-Hellman 算法

在基本的 Diffie-Hellman 算法中，一组用户共享素数 p 和生成元 g，A 的私钥为 s，公钥为 $t=g^s \bmod p$。

下面是公正的 Diffie-Hellman 算法(本例中采用5位委托人)。

(1)A 选取5个都小于 p 的整数 s_1,s_2,s_3,s_4和 s_5，A 的私钥为：

$$S=(s_1+s_2+s_3+s_4+s_5) \bmod p$$

公钥为

$$T=g^s (\bmod p)$$

A 还计算

$$t_i=g^{si} (\bmod p) \qquad i=1,2,\cdots,5$$

A 的公开一份为 t_i，私自一份为 s_i。

(2)A 给每位委托人发送一个 s_i 及相应的 t_i，例如，将 s_1和 t_1送给委托人1，把 T 送给 KDC。

(3)每位委托人验证

$$t_i=g^{si} (\bmod p)$$

如果成立，委托人签名 t_i 并送给 KDC(公钥分布中心)，接着委托人安全地存储 si。

(4)接收到所有的 t_i 之后，KDC 验证

$$T=(t_1 * t_2 * t_3 * t_4 * t_5) \bmod p$$

如果成立，KDC 赞同公钥。

这时，KDC 知道每位委托人有一部分有效的 s_i，并且如果需要他们可以重构私钥，然而，无论是 KDC 还是单个委托人都不能重构 A 的私钥。

S. Micali 还讨论了公正的 RSA 算法，以及把阈值算法和好的密码系统组合的方法，使得 n 位委托人中的 m 位能够重构私钥。

16.11　或全或无的秘密揭示

由 A. Salomaa 和 L. Santean 设计的这个协议可以使多方(至少二个)从同一人中购买某一秘密。

首先现有二个位串 x 和 y，定义 x 和 y 的固定位索引（fixed-bit index，FBI）为 x 与 y 的对应位相同的各位。

例如：

x＝110101001011

y＝101010000110

FBI(x，y)＝｛1，4，5，11｝（从右到左扫描，最右边为第0位）

在本协议中，假设 A 为出售者，B 和 C 为购买者，A 有 k 个 n 位的秘密：S_1，S_2，…S_k，B 想要购买 S_b，C 要购买 S_c。

(1)A 生成一个公钥/私钥对并把公钥告诉 B（不告诉 C），她生成另一个公钥/私钥对并把公钥告诉 C（不告诉 B）。

(2)B 生成 k 个 n 位的随机数 B_1，B_2，…B_k，并把它们告诉 C，C 生成 k 个 n 位的随机数 C_1，C_2，…C_k，并把它们告诉 B。

(3)B 用来自 A 的公钥加密 C_b（他要购买 S_b），他计算 C_b 与加密结果的 FBI，并把 FBI 送给 C。C 用来自 A 的公钥加密 B_c（他要购买 S_c），他计算 B_c 与加密结果的 FBI，并把 FBI 送给 B。

(4)B 把每个 n 位数 B_1，B_2，…B_k 中索引不在来自 C 的 FBI 的各位取补，并把新的 n 位数 $B_{1'}$，$B_{2'}$，…$B_{k'}$送给 A。C 把每个 n 位数 C_1，C_2，…C_k 中索引不在来自 B 的 FBI 的各位取补，并把新的 n 位数 $C_{1'}$，$C_{2'}$，…$C_{k'}$送给 A。

(5)A 用与给 B 的公钥相应的私钥解密所有的 $C_{i'}$，得到 k 个 n 位的数 $C_{1''}$，$C_{2''}$，…$C_{k''}$，计算 S_iXORC_i"(i＝1，2，…，k)，并把结果送给 B。A 用与给 C 的公钥相应的私钥解密所有的 $B_{i'}$，得到 k 个 n 位的数 $B_{1''}$，$B_{2''}$，…$B_{k''}$，计算 SiXOR$B_{i''}$(i＝1，2，…，k)，并把结果送给 C。

(6)B 把 Cb 与第 b 个来自 A 的数异或而得 S_b。C 把 Bc 与第 c 个来自 A 的数异或而得 Sc。

例如：A 有下列8个12位的秘密：S_1＝1990，S_2＝471，S_3＝3860，S_4＝1487，S_5＝2235，S_6＝3751，S_7＝2546，S_8＝4043，B 要购买 S_7，C 要购买 S_2。

(1)A 采用 RSA 算法，她与 B 使用的密钥对为：n＝7387，e＝5145，d＝777。她与 C 使用的密钥对为：n＝2747，e＝1421，d＝2261。她把各个公钥分别告诉 B 和 C。

(2)B 生成8个12位的随机数：B_1＝743，B_2＝1988，B_3＝4001，B_4＝2942，B_5＝3421，B_6＝2210，B_7＝2306，B_8＝222，并告诉 C。C 生成8个12位的随机数：C_1＝1708，C_2＝711，C_3＝1969，C_4＝3112，C_5＝4014，C_6＝2308，C_7＝2212，C_8＝222，并告诉 B。

(3)B 要购买 S_7，于是他用 A 给他的公钥加密 C_7：

2212^{5145}(mod 7387) ＝5928

于是：

2212＝0100010100100

5928＝1011100101000

这二个数的 FBI 为｛0，1，4，5，6｝，送给 C。

同理，C 加密 B_2，并计算出 B_2与加密结果的 FBI 为｛0，1，2，6，9，10｝，送给 B。

(4)B 把 B_1，B_2，…B_8中索引不在｛0，1，2，6，9，10｝中的各位取补，例如：

B_2＝111111000100＝1988

$B_{2'}$＝011001111100＝1660

把 $B_{1'}$，$B_{2'}$，…$B_{8'}$送给 A。

C 把 C_1,C_2,…C_8中索引不在{0,1,4,5,6}中的各位取补,例如:

C_7=0100010100100=2212

$C_{7'}$=1011100101000=5928

把 $C_{1'}$,$C_{2'}$,…$C_{8'}$送给 A。

(5)A 用与 B 相应的私钥解密所有的 $C_{i'}$,并将结果与 S_i 异或,例如,对于 i=7:

5928^{777}(mod 7387)=2212

2546 XOR 2212=342

把结果送给 B。

A 用与 C 相应的私钥解密所有的 $B_{i'}$,并将结果与 S_i 异或,例如,对于 i=2:

1660^{226} (mod 2747)=1988

471 XOR 1988=1555

把结果送给 C。

(6)B 把 C_7与第7个来自 A 的数异或而得 S_7:

2212 XOR 342=2546

C 把 B_2与第2个来自 A 的数异或而得 S_2:

1988 XOR 1555=471

这个协议可以有任意多个购买者,如果 B,C 和 D 要购买秘密,A 给每位购买者二个公钥(每位其他人一个),每位购买者从其他购买者得到一些数,最后,把他们最后所有来自 A 的结果异或而得他们的秘密。

16.12 秘而不宣的知情证明

离散对数的秘而不宣的知情证明

假设 P 要向 V 证明她知道满足下面方程的 x:

$A^x = B(\text{mod } P)$

其中 p 为素数,x 与 P-1互素,A,B 和 P 数是公开的,x 随机选取。下面协议说明不暴露 x,P 如何证明她知道 x:

(1)P 生成 t 个均小于 p-1的随机数 $r_1, r_2, \cdots r_t$。

(2)对于所有的 r 值,P 计算 $h_i = A^{r_i}(\text{mod } p)$,并把结果送给 V。

(3)P 和 V 执行一个掷币协议以生成 t 位:$b_1, b_2, \cdots b_t$。

(4)对于所有的 t 值,P 执行下列之一:

(a)如果 $b_i=0$,则把 r_i 送给 V_r。

(b)如果 $b_i=1$,则把 $s_i = r_i - r_j(\text{mod } p-1)$送给 V。

其中 j 为 $b_i=1$的 i 的最低值。

(5)对于所有的 t 值,V 确认下列之一:

(a)如果 $b_i=0$,则确认 $A^{r_i}=h_i$。

(b)如果 $b_i=1$,则确认 $A^{s_i}=h_i h_j^{-1}$。

(6)P 把 Z 送给 V,其中:

$Z = x - r_j(\text{mod } p-1)$

(7)V 验证:

$A^Z = B * h_j^{-1}$

P 行骗的可能性为$1/2^t$。

改进的离散对数的秘而不宣的知情证明

类似于前面的协议，A，B 和 P 数是公开的，P 要向 V 证明她知道满足下面方程的 x：

$$A^{x}=B(\bmod P)$$

协议如下：

(1)P 选择小于 p－1的随机数 r，计算 $h=A^r$，并把 h 送给 V。

(2)V 把一个随机位 b 送给 P。

(3)P 把下式计算而得的 s 送给 V：

$$s\equiv r+bx(\bmod p-1)$$

(4)V 验证：

$$A^{s}=hB^b$$

(5)重复(1)到(4)t 次。

P 欺骗的可能性为$1/2^t$。

这个证明可以修改为模数 n 为合数，其中 n 为二个素数 p 和 q 的乘积，在第(1)步中，P 选择一个小于(p－1)*(q－1)的数，在第(3)步中，P 计算 $s=r+bx(\bmod((p-1)*(q-1)))$，并把 s 送给 V。

攻破 RSA 能力的秘而不宣的知情证明

假设 A 知道 C 的私钥，A 要让 B 相信她知道 C 的私钥，然而，她不想把密钥告诉 B 或者为 B 把 C 的某报文解密，协议如下：

设 C 的公钥为 e，私钥为 d，RSA 的模数为 n。

(1)A 和 B 取相同的随机数 k 和 m，使得：

$$km=e$$

他们应随机选取这些数，用任何的掷币协议生成 k 并计算 m，如果 k 和 m 都大于3，协议继续执行，否则重新选择。

(2)A 和 B 生成一个随机密文 C，他们得再次使用掷币协议。

(3)A 采用 C 的私钥计算：

$$M=C^d \bmod n$$

再计算：

$$X=M^k \bmod n$$

并把 X 送给 B。

(4)B 验证：

$$X^m \bmod n=C$$

如果成立，他相信 A。

16.13 盲签名

盲签名(blind signature)的思想由 D. Chaum 设计，它采用 RSA 算法。

假设 B 有公钥 e，私钥 d，以及公开的模数 n，A 要 B 在他未见的报文 m 上签名。

A 选择一个在1到 n 之间的随机数 k，用计算

$$t\equiv mk^e(\bmod n)$$

掩盖了 m。

B 在 t 上签了名：

$$t^d=(mk^e)^d(\text{mod } n)$$

A 计算

$$s=t^d/k(\text{mod } n)$$

而恢复签名。因为：

$$t^d=(mk^e)^d= m^{dk}(\text{mod } n)$$

所以

$$t^d/k=m^{d*}k/k=m^d$$

因此结果为：

$$s=m^d(\text{mod } n)$$

Chaum 还设计了一系列更复杂的盲签名算法。

16.14 茫然传送

在 Rabin 设计的这个协议中，A 有50%的机会把二个素数 p 和 q 送给 B，A 不会知道传送是否成功。

(1)A 把二个素数的乘积 $n=pq$ 送给 B。

(2)B 选择一个小于 n 并与 n 互素的随机数 x，把

$$a=x^2\text{mod } n$$

送给 A。

(3)A 知道 p 和 q，计算 a 的4个根：x，n－x，y 和 n－y，随机选取一个根送给 B。

(4)如果 B 接收到 y 或 n－y，他能计算 x＋y 和 n 的最大公因子，要么是 p，要么是 q，接着当然 $n/p=q$。

如果 B 接收到 x 或 n－x，他不能计算出什么东西。

16.15 安全的多方计算

假设 A 知道整数 i，B 知道整数 j，A 和 B 一起想知道是 i＞＝j 还是 i＜j，但又不希望暴露这些整数。

本例中假设 i 和 j 的范围为1到100，B 有一个公钥和一个私钥。

(1)A 选取一个大的随机数 x，并且用 B 的公钥加密：

$$c=E_B(x)$$

(2)A 计算 c－i 并把结果送给 B。

(3)B 计算下列100个数：

$$y_u=D_B(c-i+u) \qquad 1<=u<=100$$

其中 DB 为用 B 的私钥的解密算法。

他选取一个大的随机数 p(p 要比 x 小一点，B 不知道 x，但 A 可告诉他 x 的大小)，他再计算下列100个数：

$$z_u=(y_u \text{ mod } p) \qquad 1<=u<=100$$

对于所有不等于 v 的 u，验证：

$$|z_u-z_u|>=2$$

以及对于所有的 u，验证：

$$0 < z_u < p-1$$

如果不成立,B选择另一个素数并重新试探。

(4)B依次把下面序列送给A:

$$z_1, z_2, \cdots, z_j, z_{j+1}+1, z_{j+2}+1, \cdots, z_{100}+1, p$$

(5)A检验序列中第i个数是否模p同余于x,如果是,他得出结论$i <= j$,否则认为$i > j$。

(6)A把结论告诉B。

在第(3)步中的各种验证是为了保证第(4)步中的序列里没有一个数出现二次,否则,如果$z_a = z_b$,则A知道$a <= j < b$。

这个协议的缺点是A比B先知道计算结果,无法阻止A在第(5)步之前结束协议并且在第(6)步中拒绝把结果告诉B,甚至在第(6)步中B可以说谎。

例子

假设采用RSA公钥算法,B的公钥为7,私钥为23,n=55,A的秘密值i为4,B的秘密值j为2(假设i和j只能取1,2,3或4)。

(1)A选取$x=39$,$c=E_B(39)=19$。

(2)A计算$c-i=19-4=15$,把15送给B。

(3)B计算下列4个数:

$$y_1 = D_B(15+1) = 26$$

$$y_2 = D_B(15+2) = 18$$

$$y_3 = D_B(15+3) = 2$$

$$y_4 = D_B(15+4) = 39$$

选取一个大的$p=31$,并计算

$$z_1 = (26 \bmod 31) = 26$$

$$z_2 = (18 \bmod 31) = 18$$

$$z_3 = (2 \bmod 31) = 2$$

$$z_4 = (39 \bmod 31) = 8$$

执行各种验证,保证序列正常。

(4)B依次把下面序列送给A:

$$26, 18, 2+1, 8-1, 31 = 26, 18, 3, 9, 31$$

(5)A检验序列中第4个数是否模p同余于x,由于9不等于39 mod 31,故$i > j$。

(6)A把结果告诉B。

16.16 概率加密

概率加密(probabilistic encryption)的思想是由S. Goldwasser和S. Micali提出的,尽管它的理论使它成为已知最安全的密码系统,但是它早期的实现并不实用,最近的实现改变了这一状况。

概率加密的要点是消除公钥密码系统泄露的最后一点信息,由于密码分析者总归可以用公钥加密随机报文,他们可以得到一些信息。假设他们有密文$C=E_K(P)$并试图恢复明文P,他们可选取一个随机报文P'并加密:$C'=E_K(P')$,如果$C'=C$,他们猜中了明文,否则也只需再猜一次。

并且,没有泄露有关原报文的部分信息,在公钥密码系统中,密码分析者有时可从位信息

中得到一些信息，例如第5，17和39位异或为1等等，而在概率加密中，即使这类信息也得到了隐藏。

尽管没有太多的新信息，让密码分析者用公钥加密随机报文还是有潜在的问题，有的信息是每次密码分析者加密报文时泄露出去的，虽然没人知道到底有多少，但也的确存在。

概率加密试图消除这些泄露，其目标不计算密文、或者在试探明文时没有给密码分析者提供相应明文的信息。

在概率加密中，加密算法更具概率性，即相应于给定的明文有许多密文，用于任何给定加密中的特殊密文是随机选取的，所有这些密文都解密为同样的明文。

$$C_1 = E_K(P), C_2 = E_K(P), C_3 = E_K(P), \cdots Ci = E_K(P)$$

$$P = D_K(C_2) = D_K(C_2) = D_K(C_3) = \cdots = D_K(C_i)$$

在概率加密中，密码分析者再也不能加密多个随机明文而寻找正确者了，例如，假设密码分析者有密文 $C = E_K(P)$，即使他们猜中了 P，当他们加密 $E_K(P)$时，得到的结果也完全不同于 C：C_j，他们无法比较 C_i 和 C_j，因此他们无法知道他们是否猜中了报文。

即使密码分析者有公开的加密密钥，明文和密文，如果没有私人的解密密钥，他们也不能证明这个密文为明文的加密结果，即使他们试图穷举搜索，他们也只能证明每个可想象的明文是可能的明文。

这类方法产生的密文总归比明文大，这是不可避免的，这是许多密文解密为相同明文的结果。第一个概率加密算法生成的密文比明文大到以致于没法使用。

然而，M. Blum 和 S. Goldwasser 有效地实现了概率加密，它采用以前讨论的 Blum Blum Shub（BBS）随机位发生器。

BBS 发生器基于平方余数理论，换言之，有二个模4同余于3的素数 p 和 q，这是私钥，它们的乘积 n＝pq 为公钥（注意 p 和 q，这个算法的安全性在于因子分解 n 的难度）。

加密报文 P 时，首先选取一个与 n 互素的随机数 x，再计算：

$$x_0 = x^2 \bmod n$$

把 x_0作为 BBS 伪随机位发生器的种子，并一次一位地把 P 和发生器的输出异或。发生器输出 b_i，则

$$P = P_1, P_2, P_3, \cdots, P_t$$

$$C = P_1 \text{XOR } b_1, P_2 \text{XOR } b_2, P_3 \text{XOR } b_3, \cdots, P_t \text{XOR } b_t$$

其中 t 为明文的长度

再在处理结束时把 x_t 附加在报文的末尾。

把报文解密的唯一方法是找到 x_0并建立其同样的 BBS 发生器以与密文异或，由于 BBS 发生器向左安全，x_t 值对于密码分析者是没有用的。只有知道 p 和 q 的人才能把报文解密。

从 x_t 中找到 x_0的 C 算法如下：

```
/ * pow(x,y)returns x to the power of y. * /
int x0(int p,int q,int n,int t,int xt)
{
    int a,b,u,v,w,z;
    /* we already know that gcd(p,q)==1 */
    (void)entended_euclidian(p,q,&a,&b);
    u=pow((p+1)/4,t)%(p-1);
```

```
    v=pow((q+1)/4,t)%(q-1);
    w=pow(xt,u)%p;
    z=pow(xt%q,v)%q;
    return(b*q*w+a*p*z)%n;
    }
```

一旦得到了 x_0,解密是容易的,只要建立 BBS 发生器并将输出与密文异或。

使用 x_i 的所有的安全位,而不仅仅是最低位可以加快算法的执行过程,这样,Blum-Goldwasser 概率加密算法比 RSA 快,并且不会泄露有关明文的部分信息。另外,可以证明攻破这个算法的难度与因子分解 n 的难度相同。

另一方面,这个算法对选择密文攻击是完全不安全的,从正确平方余数的最低位是可以计算出任何平方余数的平方根的,这样就可以作因子分解了。

16.17 量子密码术

量子密码术(quantum cryptography)引自量子世界的自然不确定性,可以用它建立一条通讯信道,在这条信道中,不干预传送是无法窃听的。物理原理保护了量子信道——即使窃听者能为所欲为,即使窃听者具有无限的计算能力,即使 P=NP。C. Bennett 和 G. Brassard 已经尽量采用这个思想,描述了量子密钥分发,量子掷币和量子承诺。

虽然量子密码术还是个争论的话题,但 Bennett 和 Brassard 构造了一个工作模型,现在有了实验性的量子密码术。

按照量子学观点,粒子不存在某一确定的位置,它们以不同概率出现于多处。一位科学家测量到粒子的精确位置,他就不能测量出其它方面如速度等量。位置和速度二个量中的任一量的测量行为都会破坏对另一量测量的可能性。量子世界的测不准关系(又称不相容性)是不可避免的。

这个不相容性可用来生成私钥。光子移动时振荡,这个振荡是具有一定的方向性的:上下,左右,或可能沿某个角度。一般的太阳光是非极化的,即光子均等地沿各个方向振荡。当一群光子沿某个方向振荡时,他们就极化了。极性过滤器只允许在某个方向极化的光子通过,例如,水平的极性过滤器只允许水平极化的光子通过,把过滤器转90度,则只有垂直极化的光子能通过。

假设有一个水平极化的光子脉冲,对水平的极化过滤器,它们都能通过。慢慢地将过滤器转至90度,通过的光子数越来越少,直到没有光子通过。这凭直觉难以理解,因为光子是水平极化的,过滤器转角甚小就会阻碍所有光子通过,但这仅仅是常识,而不是量子力学的意义,每个粒子有突然转换其极性以与过滤器匹配的概率,如果角度只是偏离一点点,则概率极高,如果偏差90度,则概率为0。如果偏差45度,则有50%的机会。下面就用这个量子特性生成私钥。

(1)A 把一串光子脉冲送给 B,每个脉冲随机极化为以下四个方向之一:水平,垂直,左对角和右对角。

例如,A 送给 B:

||/——\—|—/

(2)B 有一个极性检测器,他可设置检测器测量水平和垂直的极性(用"+"表示),或者测量对角极性(用"X"表示),但不能同时测量二者,测量一种极性就不可能测量另一种。他随意设置检测器,例如:

X＋＋XXX＋X＋＋

于是，当B正确地设置检测器时，他就能记录下正确的结果。如果他设置检测器测量水平或垂直方向，并且脉冲极性要么是水平的，要么是垂直的，他就可以知道A把光子如何极性化的。如果检测器设置不正确，他会得到随机的结果而无法区分。本例中他可能得到如下的结果：

||/——\—|—/

(3)B通过不安全的信道告诉A他的设置方式。

(4)A告诉B哪些设置方式是正确的。本例中，第2，6，7和9个脉冲设置是正确的。

(5)A和B只保留正确的设置方式，本例中他们保留：

* | * * * \ - * - *

A和B用预先安排的编码各自把这些设置方式转换为二进制位，例如，水平和左对角为1，垂直和右对角为0。本例中他们都得到4位结果：

0 0 1 1

可以用这个系统生成任意位。平均情况下，B能猜中50%的设置方式，因此A要发送2n个光子脉冲才能生成n位。他们可以用这些二进制位作为常规算法的私钥，或作高安全的一次一密之用。

这个系统不易窃听。如果窃听者检测经过的光子，B就会发觉，窃听者唯一的希望只是捕捉并测量光子，再把相同者送给B。但象B一样，要猜测测量哪种极性的光子，并且与B一样，有一半的猜测是错误的，无法控制在送给B的脉冲中发生错误。

如果窃听者这样做了，A和B会得到不同的位串，于是，他们这会如此结束协议：

(6)A和B比较部分他们的位串，如果不一致，他们就知道有人在窃听，如果完全相同，他们丢弃用于比较的那部分，并使用剩余的二进制位。

虽然以上仅仅是个工作模型，英国的Telcom已在10km的光纤联结上做了试验。

第四部分 现实世界

第17章 实现实例

为密钥管理、认证和加密而设计协议和算法是一回事，但是在可操作的系统内实现它们则又是另一回事，貌似良好的思想往往不能实际运作，其原因也许是由于带宽要求太大，也许是由于协议中延时太长。

下述各密钥管理协议中大部分都已实现。

17.1 IBM 私钥管理协议

这是计算机网络中保证通讯和文件安全的完整的密钥管理系统，并只采用对称密码术，它是70年代末由 IBM 公司的 Ehrsam，Matyas，Meyer 和 Tuchman 设计的。

这个协议有三个主要特征：服务器与多台终端之间的安全通讯、服务器中安全的文件存储以及服务器之间的安全通讯。它并不真正提供终端之间的直接通讯，须修改后实现。

网络上每台服务器与一密码设备相连，由密码设备完成所有的加密和解密工作。每台服务器有一个主钥(master key)KM0以及二个变体 KM1和 KM2，KM1和 KM2都是 KM0的简单变体，这些密钥用于加密其它密钥以及生成新的密钥。每台终端有一个主终端密钥(master terminal key)KMT，用于便利不同计算机之间的密钥交换。

服务器存储 KMT，并由 KM1加密。所有其它密钥，例如用于加密密钥文件(称为 KNF)的密钥，都用 KM2加密存储。主钥 KM0存储在非挥发性的安全模块中，例如可以成为 ROM 密钥或磁卡，或者由用户手工输入计算机(也许作为一个文本串再作密钥裂变)。KM1和 KM2不存储在系统中，但可在需要的时候从 KM0计算出来，服务器之间通讯所需的会话密钥是在服务器中用伪随机程序生成，用于加密存储文件(KNF)的密钥也以同样的方法生成。

协议的核心是一个称为密码设施(cryptographic facility)的防篡改模块，在服务器和终端里，所有的加密和解密操作都在这个设施内完成。最重要的密钥——用于生成实际加密密钥的密钥也都存储在这个模块中，这些密钥一旦存储之后就再也不能读出，Davies 和 Price 详细讨论了这个密钥管理协议。

变例

这个主钥和会话密钥算法有一种变例，这个算法是环绕网络结点与为本地终端服务的密钥公证设施而建立的，它提供下列功能：

·任二个终端用户之间的安全双向通讯

·用加密邮件安全通讯

·保护私人文件

·提供数字签名

为了各用户之间的通讯和文件传送，这个算法的密钥是在密钥公证设施内生成，并在主钥加密下传送给各用户。用户的身份与密钥结合，证实对用户之间已在用的会话密钥。尽管这个

系统没有使用公钥密码术，但具有类似数字签名的功能，密钥只能来自某一源处，且只能在某个目的地才能读出来。这就是密钥公证的特点，它是系统的中心。

17.2 MITRENET

公钥密码术的最早实现之一是实验性的系统 MEMO(MITRE 加密邮件办公室),MEMO是 MITRENET 网络上安全的 E-Mail 系统，其中密钥交换采用公钥密码术，而文件加密采用DES。

在 MEMO 系统中，公钥分发中心(Public Key Distribution Center)是网络上一个独立的结点，它有所有的公钥，并存储在 EPROM 中以防止任何改动。私钥由各用户或者系统生成。

用户每次入网时首先建立与公钥分发中心的安全通讯线路，用户需要一个由中心来的所有公钥的文件，如果用户用他们自己的私钥通过了身份测试，中心就把公钥表送给每个用户的工作站，为保证文件完整性，这个公钥表是用 DES 加密的。

当用户要给别的用户发送报文时，他们首先生成一个随机的 DES 密钥(其实由系统本身完成)，他们再用 DES 密钥加密文件，并且用接收者的公钥加密 DES 密钥，把用 DES 加密的文件和用公钥加密的密钥都送给接收者。

MEMO 对丢失的密钥不作处理，但提供了校验和检验报文完整性的功能，系统内也没有认证机制。

尽管采用大数可修改这个系统，系统在实现之前，已经证明用于这个系统的某个公钥(在 GF(2127)上的 Diffie-Hellman 密钥交换)是不安全的，MEMO 主要是为实验目的而设计的，并从未在真正的 MITRENET 系统上运行。

17.3 ISDN

Bell-Northern 研究组开发了一种实验性的、安全的综合业务数字网(ISDN)终端，这种终端可以用64kbps 的速度发送和接收声音及数据，它采用 Diffie-Hellman 密钥交换协议，RSA 数字签名以及 DES 数据加密。

密钥

电话中嵌有一个长期的公钥/私钥对，私钥存储在电话内防篡改的地方，公钥作为电话的标识符，这些密钥是电话的一部分并且不能改变。

另外，还有其它二个公钥存储在电话机中，其中之一为拥有者的公钥，它用于认证来自拥有者的命令，拥有者通过签发一个命令可以改变这个密钥，这样，拥有者可以把电话的所有权移交给他人。

电话中存储的另一个公钥为网络的公钥，它用于认证来自网络密钥管理设施的命令，以及认证网络上其他用户的呼叫，拥有者通过签发一个命令也可以改变这个密钥，这样，拥有者可以把电话从一个网络移到另一个网络。

这些密钥认为是长期的密钥，因为它们很少会改变。电话中还存储有一个短期的公钥/私钥对，它们压缩在一个由密钥管理设施签名的证书中，当二台电话机通话时，它们交换证书。网络的公钥用于认证这些证书。

证书交换和验收只建立了电话之间的安全呼叫，要建立用户之间的安全呼叫还有另外一部分协议。拥有者的私钥存储在硬件"点火密钥"上，该点火密钥由拥有者插入到电话机中，点火密钥中有拥有者的私钥，并且私钥用只有拥有者知道的秘密口令加密(电话机、网络密钥管

理设施和其它人都不知道)。点火密钥中还有一个由网络密钥管理设施签名的证书,该设施内有拥有者的公钥和一些身份标识信息(姓名、公司、安全许可证、职称等等),这也是经过加密的。要把这些信息解密并输入电话机中时,拥有者必须在电话机键盘上输入他的秘密口令。在电话机用这些信息呼叫之后,拥有者去除点火密钥时这些信息也就销毁了。

电话机中还存储有一组来自网络密钥管理设施的证书,这些证书允许特定的用户使用特定的电话机。

呼叫

从 A 到 B 的呼叫过程如下:

(1)A 把点火密钥插入电话机并输入口令。

(2)电话机验证点火密钥以确定 A 的身份。

(3)电话机检验证书,以保证该用户允许使用本电话机。

(4)A 拨号,电话机发出呼叫。

(5)二台电话机用一个公钥密码术密码交换协议生成一个唯一且随机的会话密钥,所有后续协议步骤都用这个密钥加密。

(6)呼叫电话机发送其证书和用户认证。

(7)被呼叫电话机用网络公钥认证证书和用户认证中的签名。

(8)B 的电话机初始化一个"询问-应答"序列,它对与时间有关(time dependent)的询问要求作出实时的并且签了名的应答(这是为防止敌手使用从前一次交换中复制的证书)。一次应答必须以 A 电话机的私钥签名。另一次则必须以 A 用户的私钥签名。

(9)B 的电话机响铃。

(10)如果 B 在场,他把点火密钥插入电话机,类似于第(2)步和第(3)步,电话机验证点火密钥和检验证书。

(11)A 的电话机发送其证书和用户认证。

(12)像第(7)步,A 的电话机认证 B 的签名,并且像第(8)步那样初始化一个"询问-应答"序列。

(13)二台电话机都显示其他用户和电话机的身份标识。

(14)安全对话开始。

(15)当一方挂断电话,像被呼叫的电话机从呼叫电话机接收到的证书以及呼叫电话机从被呼叫电话机接收到的证书一样,会话密钥就删除了。

每次呼叫时的 DES 密钥都是唯一的,它只在呼叫期间在二台电话机内存在并在呼叫之后立即销毁。如果敌手捕捉到呼叫中的一台电话机,或者甚至二台电话机都捕捉到,敌手也不能将二台电话机之间的前面呼叫解密。

17.4 KERBEROS

Kerberos 是为 UNIX TCP/IP 网络设计的三方认证协议(Kerberos 一词意为神话中的三头警犬),网络上的 Kerberos 服务设施作为信赖的仲裁者,Kerberos 提供安全的网络认证,允许一个用户访问网络上的不同机器。Kerberos 基于对称密码术(实现时采用 DES,但也可以用其它算法),Kerberos 与网络上每一个实体共享一个不同的私钥,私钥的知识用作身份证据。

Kerberos 认证服务设施最初在 MIT 为雅典娜计划而设计。UNIX 软件已放在公共域并人人均可使用,Kerberos 模型部分基于 Needham 和 Schroeder 三方协议以及基于 Denning 和

Sacco 的修改。Kerberos 版本1和版本3为内部开发版本，后来有版本4和版本5。

Kerberos 模型

以前叙述了基本的 Kerberos 协议，在 Kerberos 模型中，网络上有客户机(client)和服务器(server)实体，客户机可以是用户，也可以是完成下列功能的独立软件程序：下卸(download)文件，发送报文，访问数据库等等。

Kerberos 作网络服务，为上面提到的可信赖的仲裁人。Kerberos 保持一个客户机及其私钥的数据库。对一个用户来说，私钥就是加密了的口令。需要认证的网络服务，象希望使用这些服务的客户机一样，用 Kerberos 登记其私钥。

由于 Kerberos 知道每个人的私钥，因此它可以建立让一个实体相信另一个实体的身份的报文。Kerberos 也建立临时的私钥，称为会话密钥(session key)，会话密钥送给一个客户机和一个服务器(或给二个客户机)，而不送给其它实体。会话密钥用于加密双方之间的报文，在这之后即销毁。

Kerberos 提供三级保护，在网络连接初始阶段提供认证，网络连接后假设所有的通讯都来自经认证的实体。接着 Kerberos 对于从一个实体向另一个实体发送的每个报文也提供认证，最后，Kerberos 对于从一个实体向另一个实体发送的每个报文同时提供认证和加密。

软件模块

Kerberos 的实现含有如下几个软件模块：

·Kerberos 应用程序库：提供应用程序客户机与服务器之间的接口，它有建立和阅读认证需求的程序，以及建立认证和加密报文的程序。

·加密库：这是 Kerberos 提供的 DES 实现，其中的 DES 源代码可用其它 DES 实现或不同的加密程序代替。Kerberos 版本4提供了一个非标准的扩散密码链接(PCBC)方式用于认证，这种方式是不安全的，在第5版中采用 CBC 方式。

·数据库：Kerberos 需要简单的数据库，每个实体一条记录，包含姓名、私钥、私钥的截止日期，以及一些管理信息。由于整个系统的安全性在于这个数据库的安全性，因此这个数据库必须要在安全的机器上运行。

·数据库管理程序：管理 Kerberos 数据库。

·管理服务器：提供数据库的读-写网络接口，用于添加、删除和更新数据库中的条目。基于完整性原因，程序的服务器侧必须在存放 Kerberos 数据库的机器上运行。当然，这台机器必须安全。

·认证服务器：提供数据库的只读网络接口，用于认证实体和生成会话密钥。由于这个服务器不修改数据库，它可在具有 Kerberos 数据库只读副本的机器上运行，这些机器也必须安全。

·数据库扩散软件：处理 Kerberos 数据库的应答。为防止 Kerberos 在所有认证需求下失效，往往最好是在多台机器上有多个认证服务器，每个服务器定期接收 Kerberos 数据库的副本。

·用户程序：用户登录 Kerberos，更改 Kerberos 口令，显示和销毁认证信息。

·应用程序。

Kerberos 如何工作?

这里讨论 Kerberos 版本5，后面会讨论版本4与版本5之间的区别。Kerberos 协议是很明确的(见图17.1)，客户机为票据许可服务(Ticket-Granting Service，TGS Ticket)从 Kerberos 服务取得票据，这张票据送给客户机，并用客户机的私钥加密。为使用某个服务，客户机要从票

据许可服务器(TGS)取得票据。假如情况良好,TGS 把票据送回客户机。客户机再把票据与认证单一起送给服务器。如果客户机的凭证没有问题,服务器允许客户机取得服务。

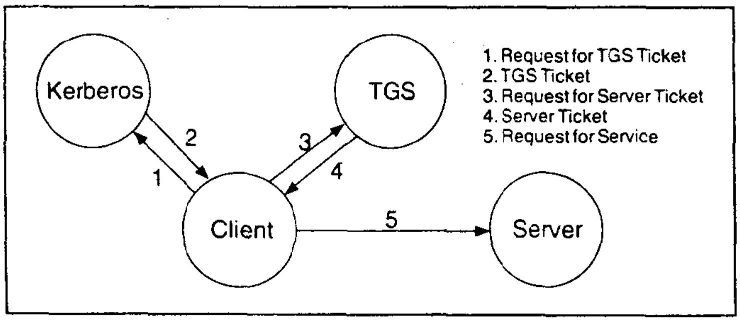


图17.1 Kerberos 认证协议

凭证

Kerberos 中使用二种凭证:票据和认证单。(下文使用 Kerberos 文档中的符号:见表17-1)。将票据发给服务器时,票据用来安全地传送客户机的身份。票据中有一些信息,服务器可用这些信息保证使用票据的客户机就是发予票据的客户机。认证单是与票据一起呈交的其它凭证。

Kerberos 票据的形式如下:

$$T_{c,s}=s,\{c,addr,times,K_{c,s}\}K_s$$

票据对单个服务器和单个客户机是有益的,它含有客户机名、服务器名、客户机的网络地址、邮戳和会话密钥,这些信息用服务器的私钥加密。客户机一旦得到这个票据,她就可以多次使用票据访问服务器,直到票据期满。客户机不能把票据解密(由于不知道服务器的私钥),但是可以以加密的形式呈交给服务器,当票据经过网络时,没有一位监听网络的人能够阅读或修改票据。

表17-1 Kerberos 缩写表

c	客户机
s	服务器
addr	客户机的网络地址
times	票据的起始和结束有效时间
Kerberos	认证服务器
Kx	x 的私钥
Kx,y	x 和 y 的会话密钥
{m}Kx	用 x 的私钥加密的 m
Tx,y	使用 y 的 x 票据
Ax.y	从 x 到 y 的认证

Kerberos 认证单的形式如下:

$$A_{c,s}=s,\{c,time,key\}K_{c,s}$$

客户机每次要在服务器上使用服务时生成认证单,它含有客户机、时戳、可选的附加会话密钥,所有都用客户机与服务器共享的会话密钥加密。与票据不同,它只能使用一次,然而,由于客户机在需要时可以生成认证单(它知道共享的私钥),这也不成问题。

认证单有二个作用,第一是它含有以会话密钥加密的一些明文,这说明认证单的发送者也知道这个密钥,并且密封的明文内含有日期时间,因此记录了票据和认证单的窃听者也不能在二天之后重发。

获取初始票据

客户机有能够证明其身份的部分信息:口令。显然我们不希望客户机通过网络传送口令,Kerberos 协议把口令泄露的机会减少到最小,同时除非某用户知道这个口令,否则不允许她认证自身。

客户机把报文送给 Kerberos 认证服务器,报文内有用户名及其 TGS 服务器名(可以有多个 TGS 服务器),现实中,用户也许只把用户名输入系统,而登录程序发送请求。

Kerberos 认证服务器在数据库中查找客户机，如果找到，Kerberos 生成客户机与 TGS 之间使用的会话密钥，这称为票据许可票据（Ticket Granting Ticket，TGT），它用客户机的私钥加密会话密钥，再为客户机建立 TGT，用以向 TGS 认证她自己，并用 TGS 的私钥加密。认证服务器把这些加密报文都送回客户机，图17.2说明了这个交换过程。

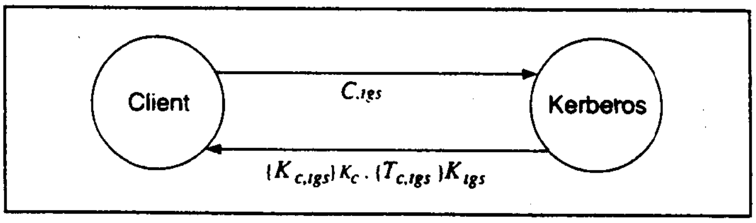


图17.2　获取 TGS 票据

于是客户机把第一个报文解密并检索会话密钥，这个私钥是用户口令的单向哈希值，因此合法用户能够顺利完成。如果用户是个冒名者，她不知道正确的口令，因此就不能将 Kerberos 认证服务器的应答解密，访问会被拒绝，因而不能得到票据或会话密钥。

客户机保存 TGT 和会话密钥，并删除口令和单向哈希值，删除这些信息是为了减少泄露机会，如果敌手复制了客户机的存储器，其结果也只是 TGT 和会话密钥。有许多部分有价值的信息，但只在 TGT 的生命期内有效，在 TGT 期满之后，它们就毫无价值。

如果每个客户机都声称他们是真实的，客户机通过 TGT 的生命期能向 TGS 证明其身份。

获取服务器票据

客户机要为每个欲使用的服务器获取一个独立的票据，TGS 为各个服务器授予票据。

当客户机需要而还没有的票据时，则向 TGS 发送请求（现实中，程序自动完成这项工作，对用户是透明的）。图17.3说明了这个交换过程。

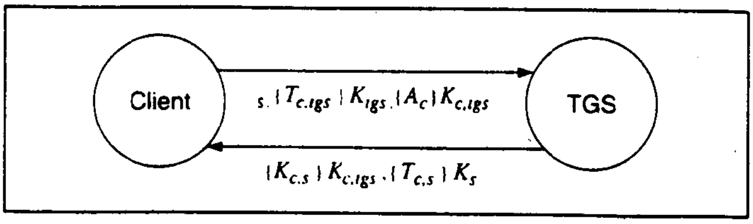


图17.3　获取服务器票据

首先，客户机建立一个认证单，认证单由客户机名、客户机地址和时戳构成，用 Kerberos 认证服务器生成的会话密钥加密。请求由服务器名、从 Kerberos 接收到的 TGT（已用 TGS 的私钥加密）以及加密的认证单构成。

TGS 接收到请求后用其私钥把 TGT 解密，再用 TGT 中包含的会话密钥把认证者解密，最后，把认证单中的信息与票据中的信息作比较，把客户机的网络地址与发送请求的地址作比较，把时戳与当前时间作比较，如果一切都匹配，则允许请求继续进行。

检验时戳时假设所有的机器都已将时钟同步，至少在几分钟以内是如此。如果请求中的时间距当前时间太远，TGS 认为这个请求是试图重发前一请求。TGS 也记录所有活的认证单，即具有时戳的以前的请求仍然有效。忽略具有相同票据和时戳、并已经接收到过的请求。

TGS 对于有效请求的响应是返回一有效票据给客户机，客户机把它交给服务器，票据中有客户机名、客户机的网络地址、时戳、票据期满时间以及刚建立的会话密钥，所有这些内容都用服务器的私钥加密，另外还有服务器名。TGS 还为客户机和服务器建立一个新的会话密钥，

用客户机和 TGS 共享的会话密钥加密。这些报文都送回给客户机。

客户机把报文解密并提取出会话密钥。

请求服务

这时客户机已准备好向服务器认证其自己，她建立一个非常类似于送给 TGS 的报文(这是有意义的，由于 TGS 是个服务)，如图17.4所示。

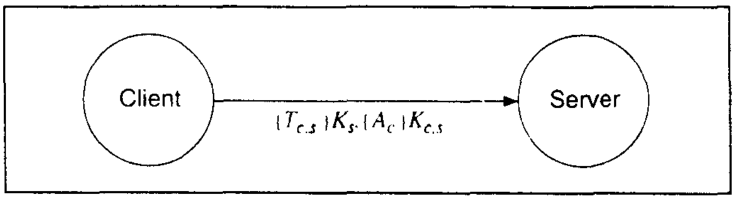


图17.4 请求服务

客户机建立认证单，认证单由客户机名、客户机的网络地址和时戳构成，并用 TGS 为客户机和服务器生成的会话密钥加密，请求由从 Kerberos 接收到的票据(已经用服务器的私钥加密)和加密的认证单构成。

如前所述，服务器解密并检验票据和认证单，也检验客户机的地址和时戳，如果一切都符合，则服务器知道客户机是真实的。

对于需要共同认证的应用场合，服务器把由时戳加一构成的报文送回客户机，这个报文用会话密钥加密(如图17.5所示)，这证明了服务器知道他的私钥并可将票据解密并作认证。

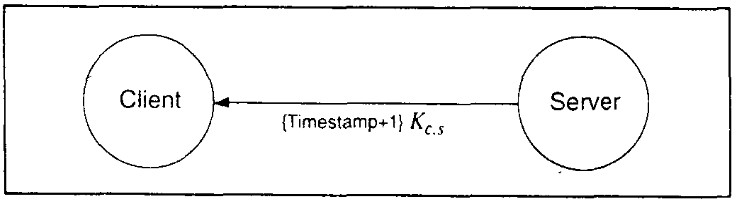


图17.5 共同认证

如果需要，客户机和服务器可用共同的密钥加密日后的报文，由于只有客户机和服务器共享这个密钥，他们都可以假设当前报文是用原先由别人生成的密钥加密的。

Kerberos 版本4

前面讨论了 Kerberos 版本5，Kerberos 版本4在报文、票据和认证单的结构上稍有不同：

Kerberos 版本4中，5个报文为：

1. 客户机———>Kerberos：c，tgs
2. Kerberos———>客户机：{Kc，tgs，{Tc，tgs}Ktgs}Kc
3. 客户机———>TGS：{Ac}Kc，tgs，{Tc，tgs}Ktgs，s
4. TGS———>客户机：{Kc，s，{Tc，s}Ks}Kc，tgs
5. 客户机———>服务器：{Ac}Kc，s}，{Kc，s}Ks

   Tc，s={s，c，addr，time，life，Kc，s}Ks

   Ac，s={c，addr，time}Kc，s

报文1，3和5是相同的，在版本5中，在第2步和第4步里的双重加密去掉了，版本5中的票据可以有多个地址，并且用一个起始时间和结束时间代替了“生命期”域，版本5中的认证单里增加了可选的附加密钥。

Kerberos 的安全性

S. Bellovin 和 M. Merritt 讨论了 Kerberos 的几个潜在的安全性弱点，尽管这篇论文是讨论版本4协议的，但许多讨论也适用于版本5。

可以储存并重发原认证单，尽管时戳技术应该防止这种情况，但在票据生命期内还是可以重发的。服务器应该存储所有的有效票据以防止重发。

认证单依赖于网络上所有的时钟或多或少是同步的，如果对主机的当前时间可以蒙骗，那么就可以毫无问题地重发原认证单，大多数的网络时间协议都是不安全的，因此这可能是个严重的问题。

Kerberos 也易受"口令猜测"攻击，入侵者可以收集票据并试图将它们解密，由于一般的用户往往不是选择好的口令，如果入侵者收集了足够多的票据，还是很可能找到口令的。

也许最严重的攻击要包括恶意的软件，Kerberos 协议依赖于 Kerberos 软件是值得信赖的，无法阻止某位入侵者除完成 Kerberos 协议之外，偷偷将所有的客户机的 Kerberos 软件替换成记录口令的软件。在不安全的计算机上，这对所有的密码术软件包都是一个问题，但 Kerberos 在这些环境里的广泛使用使它成为一个特别吸引人的目标。

17.5 KRYPTOKNIGHT

KryptoKnight 是由 IBM 设计的认证和密钥分发系统，这是一个私钥协议，并且使用 CBC 方式的 DES 或者经修改的 MD5。

KryptoKnight 支持下列四项安全性服务：

·用户认证

·双方认证

·密钥分发

·数据由来和内容认证

对于用户的角度来说，KryptoKnight 是类似于 Kerberos 的。

17.6 ISO 认证机制

公钥密码术已经推荐为与 ISO 认证机制使用，即 X. 509协议，这个机制为整个网络提供认证。虽然没有为安全性或认证指定某个算法，但是说明要求一个既能用于安全性又能用于认证的算法，并且要求加密与解密处理是互逆的（这排除了 ElGamal 和 DSA 算法，并且强烈隐指 RDA 算法），对多个算法和哈希函数也都有了规定。

证书

基于证书的这一机制，每个用户有互不相同的名字，一个可信赖的证明机构给每个用户赋予唯一的名字并发给含有名字和用户公钥的证书。

X. 509证书的形式如下：

```
certificate::=SIGNED SEQUENCE(
  signature AlgorithmIdentifier,
  issuer Name,
  validity Validity::=SEQUENCE (
notBefore UTCTime,
notAfter UTCTime)
```

subject Name,

sunjectPublicKeyInfo SubjectPublicKey::=SEQUENCE (

algorithm AlgorithmIdentifier,

subjectPublicKey BIT STRING))

证明机构(CA)签署所有证书。如果A和B要通讯,他们要相互验证对方证书的签名,如果他们使用同一个CA,这并不困难,但如果他们使用不同的CA,这就会复杂得多。考虑树结构,其中不同的CA证明其它CA及用户,根结点为主CA,每个CA存储从其父结点得到的证书。A和B要在这证明树中寻找共同信赖的结点。

如图17.6所示,A的证书由CA_A证明,B的证书由CA_B证明,CA_A的证书(要在A的证书上验证CA_A的签名)由CA_C证明,而CA_C由CA_E证明,CA_E由CA_F证明,CA_B由CA_D证明,CA_D由CA_F证明,这是个共同信赖结点并可对A和B作相互证明。

如果A和B要在网络上搜寻不同的证书时,这个过程就复杂了,现实中他们都可以存储所有的证书以加快处理过程。

证书具有特有的有效期,当证书期满时,应从由CA维护的所有共同目录中删除。发出证书的CA应保留证书的副本,在万一需要解决争论的时候用得到。

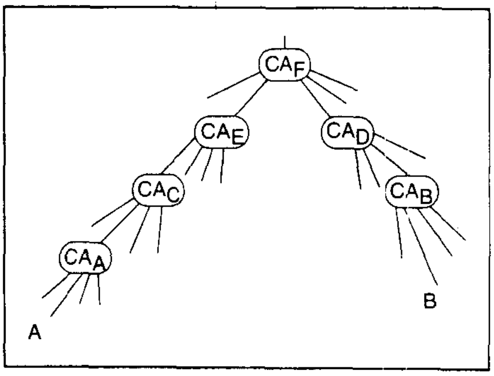


图17.6 证书层次结构

协议

A要与B通讯,首先她进入某目录并获得从A到B的"证明路径(certification path)"以及B的公钥,这时A可以初始化单向或双向或三向的认证协议。

单向协议是A向B的单向通讯,协议建立A和B的身份以及A向B通讯信息的完整性,协议也防止通讯过程中的重发攻击。

双向协议与单向协议相同,但增加B的应答。协议建立B(不是入侵者)发送应答,协议也建立通讯的保密性以及防止重发攻击。

单向协议和双向协议都使用时戳,三向协议增加了从A到B的另一个报文并且不需要时戳。

单向协议如下:

(1)A生成一个随机数R_A。

(2)A构造报文$M=(T_A,R_A,I_B,data)$,其中T_A为A的时戳,I_B为B的身份,data为任意一部分信息,并且可以用B的公钥E_A加密以提高安全性。

(3)A把$(C_A,D_A(M))$送给B。

(4)B验证C_A并得到E_A,确保这些密钥没有期满。

(5)B用E_A解密$D_A(M)$,这验证了A的签名和签名信息的完整性。

(6)B为精确性检验M中的I_B。

(7)B检验M中的T_A并确认报文正确。

(8)B依赖于老随机数的数据库检验M中的R_A,以保证这个报文不是重发的老报文。这一步可选。

双向协议由单向协议以及由从B到A的类似的单向协议构成。在执行单向协议的第(1)步到第(8)步之后,双向协议继续执行下列步骤:

(9)B 生成另一个随机数 R_B。

(10)B 构造报文 $M'=(T_B,R_B,I_A,R_A,data)$，其中 T_B 为 B 的时戳，I_A 为 A 的身份，data 为任意的，并且可以用 A 的公钥 E_A 加密。R_A 为 A 在第(1)步生成的随机数。

(11)B 把 $D_B(M')$送给 A。

(12)A 用 E_B 解密 $D_B(M')$，这验证了 B 的签名和签名信息的完整性。

(13)A 为精确性检验 M'中的 I_A。

(14)A 检验 M'中的 T_B 并确认报文正确。

(15)A 检验 M'中的 R_B，以保证这个报文不是重发的老报文。这一步可选。

三向协议完成双向协议同样的内容，但是没有时戳。第(1)步到第(15)步与双向协议相同，并且 $T_A=T_B=0$。

(16)A 依赖于在第(3)步送给 B 的 R_A 检验接收到的 R_A。

(17)A 把 $D_A(R_B)$送给 B。

(18)B 用 E_A 解密 $D_A(R_B)$，这验证了 A 的签名和签名信息的完整性。

(19)B 依赖于在第(10)步送给 A 的 R_B 检验接收到的 R_B。

17.7 增密邮件(PEM)

PEM 是 Internet 增密邮件(Privacy-Enhanced Mail)的标准，由 Internet 网络委员会 IAB (Internet Architecture Board)采用以提供 Internet 上的安全电子邮件。PEM 最初由 Internet 网络资源部 IRTF(Internet Resources Task Force)的保密性和安全性研究组 PSRG(Privacy and Security Research Group)设计，并转交给 Internet 网络工程部 IETF(Internet Engineering Task Force)的 PEM 研究组。PEM 提供加密、认证、报文完整性和密钥管理。

完整的 PEM 协议最初在一系列 RFC(Requests for Comment)中详述并后来作了修改，在第三版协议由一位 PSRG 成员 M. Bishop 作了概述，协议经过修改和改进，最后在另一系列 RFC 中作了详述，M. Bishop 详述了协议的修改。

PEM 是个广范围的标准，PEM 过程和协议意欲与广范围的密钥管理方法兼容，包括数据加密的私钥和公钥方法。对称密码术用于报文-文本加密，密码哈希算法用于报文完整性。其他文档说明基于使用公钥证书支持密钥管理机制、算法、方式、相关标识符、论文和电子格式的详情，为密钥管理基础建立的程序都在这些服务的支持之中。

PEM 最初只支持某些算法，报文以 CBC 方式的 DES 加密，由报文完整性检验 MIC(Message Integrity Check)提供的认证使用 MD2或 MD5算法，对称密钥管理使用 ECB 方式的 DES 或者使用二个密钥的三重 DES(称为 EDE 方式)。PEM 也支持为密钥管理而使用公钥证书，证书结构使用 RSA 算法(密钥长为1024比特)和 X. 509标准。

端对端密码术提供三个“保密性增强服务”：秘密性，认证和报文完整性。E-Mail 系统上不需要特殊的处理需求。PEM 可以由站点(site)或用户选择收编，不会影响网络的其它部分。

PEM 文档

有四个 PEM 文档：

·RFC1421(第一部分)，报文加密和认证过程。这个文档定义报文加密和认证程序，为 Internet 上传送的 E-Mail 提供增强保密性的邮件服务。

·RFC1422(第二部分)，基于证书的密钥管理。这个文档定义支持的密钥管理结构与基础，基于公钥证书技术把密钥信息提供给报文的发送者和接收者。

·RFC1423(第三部分),算法、方式和标识符。这个文档为密码算法、使用方式以及相关标识符和参数提供定义、格式、参考和引用。

·RFC1424(第四部分),密钥证明和相关服务。这个文档描述支持 PEM 的三种服务:密钥证明,证书取消列表(CRL)存储和 CRL 检索。

证书

PEM 是 X.509的超集,它从 TCP/IP 和 OSI 为密钥管理基础建立程序和契约,以与 PEM 和其它协议一起使用。

密钥管理基础在 Internet 内为所有证明设立一个 root:Internet 政策注册机构 IPRA(Internet Policy Registeration Authority),IPRA 设立应用到所有证明的全球政策,IPRA 的下面是各个政策证明机构 PCA(Policy Certification Authority),每个 PCA 为用户和组织登记建立和发行政策,每个 PCA 由 IPRA 证明。在 PCA 的下面设立证明机构 CA(Certification Authority),以证明用户和下级组织实体(如部门、办公室、下级部门等)。最初大部分用户要与一些组织一起登记。

有的 PCA 是为要独立登记的用户提供证明,对于要匿名使用 PEM 保密设施的用户,要与各项政策一起设立一个或多个 PCA,这些政策允许不希望揭示其身份的用户在下属 PCA 登记。

PEM 报文

PEM 的核心是其报文格式,图17.7给出用对称密钥管理的加密报文,图17.8给出用公钥密钥管理的加密报文,图17.9给出用公钥密钥管理的认证(但没有加密)报文。

```
---BEGIN PRIVACY-ENHANCED MESSAGE---
Proc-Type: 4,ENCRYPTED
Content-Domain: RFC822
DEK-Info: DES-CBC,F8143EDE5960C597
Originator-ID-Symmetric: linn@zendia.enet.dec.com,,
Recipient-ID-Symmetric: linn@zendia.enet.dec.com,ptf-kmc,3
Key-Info: DES-ECB,RSA-MD2,9FD3AAD2F2691B9A,B70665BB9BF7CBCDA60195DB94F727D3
Recipient-ID-Symmetric: pem-dev@tis.com,ptf-kmc,4
Key-Info: DES-ECB,RSA-MD2,161A3F75DC82EF26,E2EF532C65CBCFF79F83A2658132DB47

LLrHB0eJzyhP+/fSStdW8okeEnv47jxe7SJ/iN72ohNcUk2jHEUSoH1nvNSIWL9M
8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkjHUlBLpvXR0UrUzYbkNpk0agV2IzUpk
J6UiRRGcDSvzrsoK+oNvqu6z7Xs5Xfz5rDqUcMlKiZ6720dcBWGGsDLpTpSCnpot
dXd/H5LMDWnonNvPCwQUHt==
---END PRIVACY-ENHANCED MESSAGE---
```

图17.7 报文例子节略(对称情况)

第一个域是"Proc-Type",标识报文中完成的处理类型,这有三类报文,"ENCRYPTED"表示报文是经过加密和认证的,"MIC-ONLY"和"MIC-CLEAR"表示报文是签了名的,但没有加密。MIC-CLEAR 报文没有经过编码,可以用非 PEM 软件阅读,MIC-ONLY 报文在阅读之前必须转化为可读形式,因此需要 PEM 软件。PEM 报文总归是签了名的,是否加密是可选的。

下一个域"Content-Domain"与安全性无关,只是说明邮件报文的类型。

"DEK-Info"域说明关于数据交换密钥 DEK(Data Exchange Key)的信息,加密算法用于加密正文和与加密算法有关的参数,目前只指定用 CBC 方式的 DES 算法,这由"DEC-CBC"

说明。后一个域说明由 CBC 方式的 DES 算法使用的初始化矢量。将来 PEM 也许会指定其它算法，会在“DEK-Info”域及其它标识算法的域中注明。

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
Proc-Type: 4,ENCRYPTED
Content-Domain: RFC822
DEK-Info: DES-CBC,BFF968AA74691AC1
Originator-Certificate:
 MIIB1TCCAScCAWUwDQYJKoZIhvcNAQECBQAwUTELMAkGA1UEBhMCVVMxIDAeBgNV
 BAoTF1JTQSBEYXRhIFN1Y3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDzAN
 BgNVBAsTBk5PVEFSWTAeFw05MTA5MDQxODM4MTdaFw05MzA5MDMxODM4MTZaMEUx
 CzAJBgNVBAYTAlVTMSAwHgYDVQQKExdSU0EgRGF0YSBTZWN1cml0eSwgSW5jLjEU
 MBIGA1UEAxMLVGVzdCBVc2VyIDEwWTAKBgRVCAEBAgICAANLADBIAkEAwHZH17i+
 yJcqDtjJCowzTdBJrdAiLAnSC+CnnjOJELyuQiBgkGrgIh3j8/x0fM+YrsyF1u3F
 LZPVtzlndhYFJQIDAQABMA0GCSqGSIb3DQEBAgUAA1kACKr0PqphJYw1j+YPtcIq
 iW1FPuN5jJ79Khfg7ASFxskYkEMjRNZV/HZDZQEhtVaU7Jxfzs2wfX5byMp2X3U/
 5XUXGx7qusDgHQGs7Jk9W8CW1fuSWUgN4w--
Key-Info: RSA,
 I3rRIGXUGWAF8js5wCzRTkdh034PTHdRZY9Tuvm03M+NM7fx6qc5udixps2Lng0+
 wGrtiUm/oytKdinz6ZQ/aQ--
Issuer-Certificate:
 MIIB3DCCAUgCAQowDQYJKoZIhvcNAQECBQAwTzELMAkGA1UEBhMCVVMxIDAeBgNV
 BAoTF1JTQSBEYXRhIFN1Y3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDTAL
 BgNVBAsTBFRMQ0EwHhcNOTEwOTAxMDgwMDAwWhcNOTIwOTAxMDc1OTU5WjBRMQsw
 CQYDVQQGEwJVUzEgMB4GA1UEChMXU1NBIERhdGEgU2VjdXJpdHksIEluYy4xDzAN
 BgNVBAsTBkJ1dGEgMTEPMA0GA1UECxMGTk9UQVJZMHAwCgYEVQgBAQICArwDYgAw
 XwJYCsnp61QCxYykN1ODwutF/jMJ3kL+3PjYyHOwk+/9rLg6X65B/LD4bJHt05XW
 cqAz/7R7XhjYCm0PcqbdzoACZtIlETrKrcJiDYoP+DkZ8k1gCk7hQHpbIwIDAQAB
 MA0GCSqGSIb3DQEBAgUAA38AAICPv4f9Gx/tY4+p+4DB7MV+tKZnvBoy8zgoMGOx
 dD2jMZ/3HsyWKWgSF0eH/AJB3qr9zosG47pyMnTf3aSy2nB07CMxpUWRBcXUpE+x
 EREZd9++32ofGBIXaialnOgVUn00zSYgugiQ077nJLDUj0hQehCizEs5wUJ35a5h
MIC-Info: RSA-MD5,RSA,
 UdFJR8u/TIGhfH65ieewe210W4tooa3vZCvVNGBZirf/7nrgzWDABz8w9NsXSexv
 AjRFbHoNPzBuxwm0AFeA0HJszL4yBvhG
Recipient-ID-Asymmetric:
 MFExCzAJBgNVBAYTAlVTMSAwHgYDVQQKExdSU0EgRGF0YSBTZWN1cml0eSwgSW5j
 LjEPMA0GA1UECxMGQmV0YSAxMQ8wDQYDVQQLEwZOT1RBUlk-
 66
Key-Info: RSA,
 O6BS1ww9CTyHPtS3bMLD+L0hejdvX6Qv1HK2ds2sQPEaXhX8EhvVphHYTjwekdWv
 7x0Z3Jx2vTAhOYHMcqqCjA--

qeWlj/YJ2Uf5ng9yznPbtD0mYloSwIuV9FRYx+gzY+8iXd/NQrXHfi6/MhPfPF3d
jIqCJAxvld2xgqQimUzoSla4r7kQQ5c/Iua4LqKeq3ciFzEv/MbZhA--
-----END PRIVACY-ENHANCED MESSAGE-----
```

图17.8 报文例子节略(对称情况)

对于对称密钥管理的报文，下一个域是“Originator-ID-Symmetric”，这有三个子域，第一个子域用唯一的 E-Mail 地址标识发送者，第二个子域可选，标识发出交换密钥的机构，第三个子域是可选的版本/期满子域。

对称密钥管理情况的下一个域处理接收者，每位接收者有“Recipient-ID-Symmetric”和

“Key-Info”二个域，前者有三个子域，与“Originator-ID-Symmetric”域标识发送者的同样方式标识接收者。

```
—-BEGIN PRIVACY-ENHANCED MESSAGE—-
Proc-Type: 4,MIC-ONLY
Content-Domain: RFC822
Originator-Certificate:
 MIIB1TCCAScCAWUwDQYJKoZIhvcNAQECBQAwUTELMAkGA1UEBhMCVVMxIDAeBgNV
 BAoTF1JTQSBEYXRhIFN1Y3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDzAN
 BgNVBAsTBk5PVEFFSWTAeFw05MTA5MDQxODM4MTdaFw05MzA5MDMxODM4MTZaMEUx
 CzAJBgNVBAYTAlVTMSAwHgYDVQQKExdSU0EgRGF0YSBTZWN1cml0eSwgSW5jLjEL
 MBIGA1UEAxMLVGVzdCBVc2VyIDEwWTAKBgRVCAEBAgICAANLADBIAkEAwHZH17i+
 yJcqDtjJCowzTdBJrdAiLAnSC+CnnjOJELyuQiBgkGrgIh3j8/xOfM+YrsyF1u3F
 LZPVtzlndhYFJQIDAQABMA0GCSqGSIb3DQEBAgUAA1kACKr0PqphJYw1j+YPtcIc
 iW1FPuN5jJ79Khfg7ASFxskYkEMjRNZV/HZDZQEhtVaU7Jxfzs2wfX5byMp2X3U/
 5XUXGx7qusDgHQGs7Jk9W8CW1fuSWUgN4w==
Issuer-Certificate:
 MIIB3DCCAUgCAQowDQYJKoZIhvcNAQECBQAwTzELMAkGA1UEBhMCVVMxIDAeBgNV
 BAoTF1JTQSBEYXRhIFN1Y3VyaXR5LCBJbmMuMQ8wDQYDVQQLEwZCZXRhIDExDTAL
 BgNVBAsTBFRMQ0EwHhcNOTEwOTAxMDgwMDAwWhcNOTIwOTAxMDc1OTU5WjBRMQsw
 CQYDVQQGEwJVUzEgMB4GA1UEChMXU1NBIERhdGEgU2VjdXJpdHksIE1uYy4xDzAN
 BgNVBAsTBkJ1dGEgMTEPMA0GA1UECxMGTk9UQVJZMHAwCgYEVQgBAQICArwDYgAw
 XwJYCsnp61OCxYykN1ODwutF/jMJ3kL+3PjYyHOwk+/9rLg6X65B/LD4bJHtO5XW
 cqAz/7R7XhjYCmOPcqbdzoACZtIlETrKrcJiDYoP+DkZ8klgCk7hQHpbIwIDAQAE
 MA0GCSqGSIb3DQEBAgUAA38AAICPv4f9Gx/tY4+p+4DB7MV+tKZnvBoy8zgoMGOx
 dD2jMZ/3HsyWKWgSF0eH/AJB3qr9zosG47pyMnTf3aSy2nBO7CMxpUWRBcXUpE+x
 EREZd9++32ofGBIXaialnOgVUn00zSYgugiQ077nJLDUjOhQehCizEs5wUJ35a5h
MIC-Info: RSA-MD5,RSA,
 jV20fH+nnXHU8bnL8kPAad/mSQlTDZlbVuxvZAOVRZ5q5+Ej15bQvqNeqOUNQjr6
 EtE7K2QDeVMCyXsdJlA8fA==

LSBBIGllc3NhZ2UgZm9yIHVzZSBpbiBOZXN0aW5nLg0KLSBGb2xsb3dpbmcgaXMg
YSBibGFuayBsaW5lOg0KDQpUaGlzIGlzIHRoZSBlbmQuDQo=
—-END PRIVACY-ENHANCED MESSAGE—-
```

图17.9 MIC-ONLY 报文节略(对称情况)

“Key-Info”域说明密钥管理参数，这个域有四个子域，第一子域说明用于加密 DEK 的算法，由于报文中的密钥管理是对称的，发送者和接收者共享同一密钥，称为交换密钥 IK(Interchange Key)，用于加密 DEK。DEK 既可用 ECB 方式的 DES 加密(以“DES-ECB”标识)，也可用 DES 加密-解密-加密的三重 DES 加密(以“DES-EDE”标识)。第二个子域说明 MIC 算法，它可以是 MD2(以“RSA-MD2”标识)或 MD5(以“RSA-MD5”标识)。第三子域是以 IK 加密的 DEK，第四子域是 MIC，也以 IK 加密。

对于公钥(不对称)密钥管理的报文，开头部分有些不同。在“DEK-Info”域之后是“Originator-Certificate”域，这个证书遵循 X.509标准。下一个域是“Key-Info”域，它有二个子域，第一个子域说明用于加密 DEK 的公钥算法，目前只支持 RSA 算法。另一个子域是 DEK，以发送者的公钥加密，这个域可选，只是为了当邮件系统把报文返回时发送者能够将其解密。下一个域是“Issuer-Certificate”域，这是签名“Originator-Certificate”者的证书。

不对称密钥管理情况的下一个域是“MIC-Info”域，第一子域说明计算 MIC 的算法，第二

子域说明签名 MIC 的算法，第三子域由用发送者的私钥签名的 MIC 构成。

不对称密钥管理情况的后面各个域处理接收者，每位接收者有二个域，"Recipient-ID-Asymmetric"和"Key-Info"二个域，前者有二个子域，第一子域标识发出接收者的公钥的机构，第二个是可选的版本/期满子域。"Key-Info"域说明密钥管理参数，第一个子域标识用于加密报文的算法，第二个子域是以接收者的公钥加密的 DEK。

PEM 的安全性

PEM 中的 RSA 密钥为508到1024比特，这对于所有人的安全性需求都已足够长。对密钥管理协议有一种更可能的攻击方法，Mallet 可能偷走你的密钥绝不应将密钥写下来或者试图哄骗你接受一个伪造的公钥。如果人人都遵循适当的步骤，PEM 提供了密钥证明会使这种情况不可能发生，但人们往往是比较草率的。

更为阴险的攻击方法是攻击者修改在你的系统上运行的 PEM，修改后的 PEM 偷偷地将所有的邮件送给攻击者，并用他的公钥加密，它甚至可把你私钥的副本送给他。如果修改后的 PEM 运行正常，你永远不会知道 PEM 在执行什么内容。

尚无有效的方法能够阻止这类攻击，你可以使用单向哈希函数并在 PEM 代码上打"指纹"，每次运行 PEM 时都检验指纹。但是攻击者可在修改 PEM 代码的同时修改指纹代码，虽然你可在指纹代码上再打上指纹，但是攻击者也可同样修改。如果能够访问你的机器，攻击者就可暗中破坏 PEM 的安全性。

教训是如果你不信赖软件运行所在的硬件，千万不要太信赖某一软件。

RIPEM

RIPEM 是 M. Riorden 编写的一个实现 PEM 协议的程序，虽然技术上不属于公开领域，这个程序可以公开获取并可由个人、非商业领域无版税地使用，文档中包含有使用许可证。

RIPEM 采用 RSA 算法的 RSA 数据安全公司的参考实现：RSA-REF。目前 RIPEM 还不是 PEM 协议的完全实现，它没有实现认证密钥的证书。

Riorden 在编写 RIPEM 之前编了一个称为 RPEM 的类似程序，原想把这个程序作为公共 E-Mail 加密系统，为了避免专利问题，Riorden 采用 Rabin 算法，但 Public Key Partners 声称他们的专利覆盖了所有的公钥密码术并威胁要控告他，Riorden 只得停止分发 RPEM 程序。

RPEM 再也没有使用过，它与 RIPEM 不兼容，由于 RIPEM 得到了 Public Key Partners 的完全同意，因此没有理由再使用 RPEM。

17.8 报文安全性协议(MSP)

报文安全性协议 MSP(Message Security Protocol)是 PEM 的军事等价物，它由 NSA 在80年代末在安全数据网络系统(SDNS)程序下开发，它是保密 E-Mail 的与 X. 400兼容的应用层协议，它用于 Defense Message System(DMS)网络中签名和加密报文。

用于"不保密的但又敏感的"报文的初步报文安全性协议(PMSP)是 MSP 的一个版本，用于 X. 400和 TCP/IP，这个协议也称为"Mosaic"。

类似于 PEM，灵活地设计 MSP 和 PMSP 软件应用为各种安全性功能提供各种算法，包括签名，哈希散列和加密。PMSP 与 Capstone 芯片一起运作。

17.9 "密佳"电子邮件加密软件(PGP)

PGP 是 P. Zimmermann 设计的公共加密程序，其中数据加密采用 IDEA 算法，密钥管理

采用 RSA 算法(用512,1024或2048比特的密钥),并把 MD5作为单向哈希函数。PGP 也在加密之前把文件压缩。

PGP 的随机公钥使用概率素数测试器,并从测量用户击键时间间隔获得初始种子。PGP 用 ANSI X9.17附录C 中的方法生成随机 IDEA 密钥,用 IDEA 代替 DES 作为对称算法。PGP 还用哈希散列的通行短语(pass phase)而不是口令加密用户的私钥。

PGP 加密的报文具有层状结构的安全性。如果密码分析者知道接收者的密钥 ID,他们从加密的报文里唯一能得到的接收者是谁,如果报文是签了名的,只有在接收者把报文解密之后才知道是谁签名报文的。他们在验证签名之后把报文解密并知道其内容。这种方法与 PEM 相比,PEM 在没有加密的报头中留下许多有关发送者、接收者和报文的一些信息。

PGP 最有趣之处是密钥管理的分布式方法,没有密钥证明机构,所有用户生成并分发他们的公钥,用户可以签名别人的公钥,对密钥的有效性增加额外的置信度。签名别人公钥的人成为他的介绍人,当用户接收到一个新的公钥时,他们检查签名这个密钥的介绍人清单,如果某位介绍人是他们信赖的,他们就会接受这个新密钥是有效的。如果有二位介绍人是他们勉强信赖的,他们也会接受这个新密钥是有效的。这个过程是全自动的,告诉 PGP 你信赖谁?并且信赖程度如何?评价你自己的信赖级别并作处理。PGP 是精心设计和精心编码的,并在公共域中它是得到军事级别加密的最佳选择。尽管对 PGP 的专利存在过相当长时间的纠葛以及美国对与军事有关加密软件出口的限制,影响着 PGP 的推广,但就象任何优秀的东西都会不胫而走广受欢迎,如今 PGP 随着 E-Mail 的普及已变成人们常用的电子邮件"信封",保护个人隐私,大大增强了安全感。

17.10 CLIPPER

Clipper 是 NSA 设计的、使用 Skipjack 加密算法的防篡改 VLSI 芯片,每个芯片有一个特殊的密钥,这个密钥对于报文是不需要的,它只用于加密每位用户的报文密钥的副本。知道这个特殊密钥的人可以把由这个芯片保护的搭线窃听通讯解密,这保证政府人员能够执行电子监视,只有政府知道这个密钥并只有当批准时才能使用。

这个特殊的密钥拆分开并存储在二个由他人保管的密钥(key-escrow)数据库中。到目前为止,有关 Clipper 芯片的许多内容,包括由他人保管的密钥数据库都还没有定义。

Clipper 芯片是为 AT&T 商业安全的音响产品而设计的,芯片功能由 NSA 设计,逻辑由 Mykotronx 公司设计,芯片由 VLSI 公司组装。

每个芯片在出售之前独特编程,芯片编程设备把下列信息写入芯片的特殊存储器中(称为 VROM 或 VIA-Link):

·系列号,每个芯片各不相同

·单元密钥,每个芯片各不相同

·系列密钥,一系列芯片里相同

·任何专用的控制软件

在芯片中会话密钥的生成过程中,芯片执行下列动作:

(1)用单元密钥加密80比特的会话密钥,生成80比特的中间结果。

(2)把80比特结果、25比特系列号和23比特的认证模式连接在一起(共128比特)。

(3)用系列密钥加密128比特数据,生成128比特位的密码块链,它称为执法字段 LEF。

(4)LEF 至少传送给要接收的 Clipper 芯片一次。

(5)二个通讯的 Clipper 芯片一起用一字段与一个随机 IV 建立同步。

(6)一旦同步之后,二个 Clipper 芯片使用会话密钥加密/解密二个方向上的数据。

这个复杂过程的要点是允许具有独特系列号和独特单元密钥的人解密以这个芯片加密的报文,这是一个由政府嵌入到系统中的后门,不通向 Skipjack 算法,但是通向 Clipper 的密钥管理方法的后门。为防止有的人击败这个方法,如果字段 LEF 已被篡改(如修改、超级加密、替换),可以把芯片另行编程而不处于安全方式。芯片也能抵御复制,防止十分熟练的敌手。

17.11 CAPSTONE

CAPSTONE 是 NSA 开发的 VLSI 密码芯片,这个芯片象 Clipper 芯片中由他人保管密钥一样实现 Skipjack 算法。另外,Capstone 芯片包括下列函数:

·数字签名算法(DSA)

·安全哈希算法(SHA)

·通用的指数算法

·使用纯声源的通用随机数产生器

密钥交换算法(也许是 Diffie-Hellman 算法)在芯片上也是可编程的,并使用上述四个函数。



计算机密码学

作者：冯晖 来凤琪 王绍银等

现在的计算机图书实在太多,而价格昂贵,所以本站将网络上收集的电子图书整理后做成 PDF 格式，目的是给大家浏览一下图书的大要,若您认为此书不错，请支持作者,到书店购买。在电脑上看书很累、对您的眼睛也有害。

本图书版权归原作者所有,不得用于商业目的。请在 24 小时之内删除此书。

化境编程界 http://www.5xSoft.com/